ATATÜRK ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ
İŞLETME ANABİLİM DALI

Yusuf Ziya AYIK

YAZILIM GELİŞTİRMEDE NESNEYE YÖNELİK YAKLAŞIM VE NESNE TABANLI BİR ÇİZİM ARACI GELİŞTİRME UYGULAMASI

DOKTORA TEZİ

TEZ YÖNETİCİSİ

Prof. Dr. Sibkat KAÇTIOĞLU

ERZURUM - 1996

ÖZET

DOKTORA TEZİ

YAZILIM GELİŞTİRMEDE NESNEYE YÖNELİK YAKLAŞIM VE NESNE TABANLI BİR ÇİZİM ARACI GELİŞTİRME UYGULAMASI

Yusuf Ziya AYIK

Danışman: Prof. Dr. Sibkat KAÇTIOĞLU
1996, Sayfa 162

Jüri : Prof. Dr. Sibkat KAÇTIOĞLU
. . . . . .
. . . . . .

Programlama teknikleri, günümüze kadar sürekli bir gelişme göstermiştir. Donanıma yakından bağımlı, öğrenilmesi ve yazılım üretilmesi çok zor olan programlama tekniklerinden başlanarak, kuşaklar halinde gerçekleşen değişimler ile, donanımdan daha bağımsız, öğrenilmesi ve kullanılması daha kolay, çeşitli teknikler geliştirilmiştir. Bunlardan, en yaygın kullanılan ve halen en son geliştirlmiş olanı, nesneye yönelik yaklaşım tekniğidir. Dördüncü kuşak sonrası, olarak da adlandırılan bu teknikler, daha çok nesneyi esas alır ve insan zekasını taklit eder. Nesneler, belirlenmiş görevleri yerine getirmek için, çeşitli fonksiyonlarla donatılmış varlıklardır. Programlamada kullanılan nesneler, gerçek hayattaki nesnelerin davranış ve özelliklerini yansıtırlar.

Çalışmamızda, programlama dillerinin teknik gelişimi ve yazılım geliştirme sistemleri ele alındıktan sonra, nesneye yönelik yaklaşım tekniği incelenmiş, bu yaklaşımın, temel kavramları ve programlama dilleri araştırılmış ve bu alanda örnek bir uygulama geliştirilmiştir. Uygulama, geometrik şekillerin çizilmesi, işlemler yapılması ve çeşitli fontlarda yazı yazılabilmesine imkan veren, bir çizim editörüdür. Bu editör kullanılarak oluşturulan çizimler, dosyalarda saklanabilmekte, yeniden ekrana getirilebilmekte ve gerektiğinde yazıcıdan çıktı olarak da alınabilmektedir.

ABSTRACT

Ph. D. Thesis

OBJECT-ORIENTED APPROACH IN SOFTWARE DEVELOPMENT AND AN OBJECT-BASED DRAWING TOOL DEVELOPMENT APPLICATION

Yusuf Ziya AYIK

Supervisor: Prof. Dr. Sibkat KAÇTIOĞLU
1996, Page 162

Jury : Prof. Dr. Sibkat KAÇTIOĞLU
. . . . . .
. . . . . .

Programming techniques have shown a continous development so far. Beginning from the programming techniques which were difficult to learn and to produce software and closely dependent on hardware units, various techniques which are easy to use and learn and more independent than hardware have been developed by means of changes achieved throughout generations. Among these, the most commonly used and still the most recently developed one is the object-oriented approach technique. These techniques which are also named as beyond the fourth generation take the object as the base and imitate the human intellect. Objects are the entities equipped with various functions to perform determined tasks. The objects used in programming reflect the behaviours and characteristics of the objects in real life

In our study, after investigating the technical development of programming languages and software development systems, the object-oriented approach technique, the essential concepts and programming languages of this approach were studied and a sample application on this field was developed. The application is a drawing editor which enables to draw geometrical figures, to manipulate and to write in different fonts. The drawings formed by using this editor can be stored in files, can be revisualised on the screen when and can be obtained from the printer as output when necessary.

TEŞEKKÜR

Doktora çalışmalarım süresince yakın ilgi ve yardımlarını gördüğüm, tez yöneticisi hocam, sayın Prof. Dr. Sibkat KAÇTIOĞLU'na, bölüm arkadaşlarım sayın Öğr. Gör. Taner DURSUN ve sayın Yrd. Doç. Dr. M. Dursun KAYA'ya, sayın Uzman Tevhid KARACALI'ya, sayın Okutman Erhan GECİKLİ, sayın Okutman Rahmi KARADABAĞ ve sayın Okutman Süleyman BASTEM'e teşekkür ederim.

TABLOLAR DİZİNİ

ŞEKİLLER DİZİNİ

İÇİNDEKİLER

GİRİŞ

Erişilmiş bulunan teknoloji seviyesi sayesinde, ham bilgi bollaşmış ve kolay elde edilebilir olmuştur. Her ortamda bilgi bulabilmemiz mümkündür. Ancak önemli olan, bu bilgilerin organize edilebilmesidir. Göz ardı edemiyeceğimiz bir gerçek var ki, o da bildiğimiz eski yöntemlerle bu bilgi okyanusuyla başa çıkmanın hem çok zor, hem de pahalı olduğudur. Orta ve büyük ölçekli birçok firma, bilgi işlem merkezlerinde yazılım geliştirme faaliyetlerini sürdürmektedir. Ancak bu merkezlerde yapılan çalışmalar genellikle ya istenilen seviyede değildir, ya da temin edilen faydanın çok üzerinde bir maliyet ile elde edilmektedirler. Bunun en büyük nedeni, tüm program ihtiyaçlarını karşılamak üzere bir araya getirilen yazılım ekiplerinin, birbirlerinden bağımsız programları bir arada çalışır tutmak için çaba sarfetmelerine rağmen, istenilen sonuca varamamalarıdır. Bu tür çalışmalarda projeler yüksek maliyetle ve gecikmeli olarak gerçekleşmekte, gecikmeler nedeniyle zaman israfı olmakta ve maliyetler artmakta, projenin hızlandırılması durumlarında ise hatalar oluşabilmekte veya proje fonksiyonları azaltılmaktadır. Bunların sonucunda bazen de hazır projeler satın alınması yoluna gidilmektedir. Bu da, ihtiyaçlara tam cevap verme konusunda problem çıkarabilmektedir.

Bütün bu problemlerin çözümlenmesi, güvenilir bir programlama tekniğinin kullanılması ile mümkün olabilir. Bu teknik öyle olmalı ki; hem maliyeti düşük seviyede tutmalı, hem proje zamanında yetiştirilebilmeli, hem de isteklere tam olarak cevap verebilmelidir. Bunların yanısıra, geliştirilen projeler yeniden kullanılabilir kodlamalara sahip olmalı, gelişmelere açık olmalı ve ömrü kısa süreli olmamalıdır. Günümüzde bunu gerçekleştirebilecek en iyi teknik, nesneye yönelik programlama tekniğidir. Bilhassa dünyada 1980'li yıllardan sonra kullanılmaya başlanan nesneye yönelik programlama, elde edilebilen bilgileri en iyi şekilde değerlendirmede kullanabileceğimiz en etkin araç durumundadır. Kullanılmaya başlandığı yıllardan bu güne kadar, kendisini kanıtlamıştır ve üzerinde eğitim yapmaya ve proje geliştirmeye değer bir hal almıştır. Dünyadaki gelişmelere paralel olarak ülkemizde de nesneye yönelimin ağırlığı giderek artmaktadır.

Etkin bir nesneye yönelik program yazabilmek için, klasik düşünce şeklini değiştirmek gerekmektedir. Hatta bir görüşe göre, ister bilgi işlem

yöneticisi, ister programcı olsun, bugüne kadar öğrenilen her şeyin unutularak yeni bir mesleğe başlıyormuş gibi davranılması istenmektedir. Nesneye yönelim, aslında grup çalışmasını gerektiren, yeniden kullanılabilirlik ve esneklik kavramlarının ön plana çıktığı bir yaklaşımdır. Nesneye yönelim metodolojisinin anlaşılmasının zorluğu, tamamen bu kavramlarda gizlidir. Sanıldığı gibi, kodlama ve tasarım aşamalarında fazla bir güçlük söz konusu değildir. Güçlük, nesnelere yönelebilmektedir. Ardından, nesne oluşturmak ve bu nesneleri kullanmak, işin teknik yönünü oluşturur. Nesne yönelimli tasarım ve uygulama geliştirme o kadar doğaldır ki, bir kez nesne yönelimli programlamaya başladıktan sonra, yapılan işin farklı bir yöntemle yapılamıyacağı kanaatine varılabilir.

Nesneye dayalı yaklaşım, tasarım ve uygulama geliştirmede gittikçe daha popüler olmaktadır. Bunun nedeni, kullanılan nesne kavramının gerçek dünya nesneleri olması ve ilişkilerinin doğrudan bilgisayar gösterimlerine dönüştürülebilmesi için doğal bir yol sağlamalarıdır. Yapılan çalışmalar göstermiştir ki; nesneye yönelik programlama, yapısal programlama veya mantıksal programlama gibi kendine has nitelikleri olan, diğer programlama tekniklerinden farklı, yeni bir programlama metodudur. Bu nedenle yapılan çalışmada nesneye yönelik programlama, bütün yönleri ile ayrıntılı olarak incelenmeye çalışılmıştır.

Çalışma üç bölümden oluşmaktadır. Birinci bölümde, programlama dillerinin teknik gelişimi ve bir bilişim sisteminin geliştirilme aşamaları ile yöntemleri ele alınmış, aşamalar ve yöntemler arasındaki farklılıklara dikkat çekilmiştir.

İkinci bölümde, nesneye yönelimin tarihçesi verilmiş, temel kavramları, yazılım geliştirme metodları ve nesneye yönelik tasarım konuları işlendikten sonra yazılım geliştirmede kullanılan nesneye yönelik programlamanın gerekliliği ve günümüze kadar gelişimi dikkate alınarak, nesneye yönelik programlama dilleri incelenmiştir.

Üçüncü bölümde, uygulama amacıyla nesneye yönelik olarak geliştirilen çizim editörünün oluşturulabilmesi ve çalıştırılabilmesi için gerekli donanım ve yazılım özellikleri incelenerek, asgari bilgiler verildikten sonra geliştirilen çizim editörü projesi analiz edilmiş ve kullanım şekli açıklanmıştır.

Birinci Bölüm

PROGRAMLAMA DİLLERİNİN TEKNİK GELİŞİMİ VE YAZILIM GELİŞTİRME SİSTEMLERİ

Programlamanın farklı amaçlarla ve farklı özellikler göz önüne alınarak yapılmak istenmesi, çok sayıda ve çok farklı yapılarda programlama tekniklerinin ve buna bağlı olarak programlama dillerinin geliştirilmesine neden olmuştur. Programlamalarda kullanılacak tekniklerin ve programlama dillerinin belirlenmesi herhangi bir projenin geliştirilebilmesi için yeterli değildir. Bunun birlikte, belki de ondan daha önce, bu projenin hangi aşamalardan geçeceği ve hangi yöntemler kullanılarak oluşturulacağı da belirlenmesi gereken önemli konulardandır.

1.1. PROGRAMLAMA DİLLERİNİN TEKNİK GELİŞİMİ

Genel anlamda ve özel amaçlarla geliştirilmek istenilen uygulama programları, bir programlama diline ihtiyaç duyarlar. Kullanılabilecek programlama dilleri, günümüze kadar hem yapıları itibariyle, hem de kullanım amaçlarının farklılıkları nedenleriyle önemli gelişmeler göstermiş ve değişik yapılarda oluşmuşlardır. Başlangıçta kesin kuralları olan, standardize olmuş programlama dilleri, amatörce geliştirilen küçük çaplı programlar için belki yeterli olabilmiş, ancak daha sonra tecrübeli programcılar tarafından geliştirilmek istenilen uygulama programlarının profesyonel amaçlarla ve daha kapsamlı olarak düşünülmeleri, daha fonksiyonel programlama dillerine ihtiyaç duyulmasına neden olmuştur.

Başlangıçta geliştirilen programlama dilleri, daha çok uygulamanın geliştirileceği bilgisayarın özelliklerinin kullanılmasını tasarlamışlardır. Bu tür dillerin program komutları bilgisayar sistemi hangi sistem ise ve nasıl ise ona bağlı program yapılarını kapsarlar. Bu dillerin öğrenilmesi ve kullanılması oldukça fazla bilgi ve beceri gerektirir. Daha sonraları geliştirilen programlama dilleri ise bilgisayarın teknik şartlarını ve sistemini pek fazla dikkate almayan, kullanıcıların veya programcıların daha rahat ve daha kapsamlı uygulamalar geliştirmelerini sağlamayı hedef alan dillerdir.[1] Bu tür programlama dilleri, zamanla ihtiyaçların değişebileceğini ve kullanıcıların farklı teknik bilgiye sahip

---

[1] Parker, S. ve Case, T. **Management Information Systems Strategy and Action.** NewYork, McGraw-Hill, 1993, s. 251

olabileceklerini düşünerek daha esnek diller olarak hazırlanmışlardır. Bu dillerde, hem profesyonel programcıların, hem de bilgisayarla hiç ilişkisi olmayan kişilerin yazılım konularında üretken olabilmeleri amaç edinilmiştir.

Programlama dillerinin teknik gelişimi Tablo 1.1'de gösterilmektedir.

Nesil	Programlama Dili	Dönemi
1	Makina Dilli	1940-1956
2	Assembly Dili	1950'den beri
3	Yüksek Seviyeli Diller	1960 " "
4	Çok Yüksek Seviyeli Programlama Dilleri	1970 " "
5	Dördüncü Nesil Sonrası Diller	1980 " "

Tablo 1.1. Programlama Dillerinin Teknik Gelişimi

1.1.1. Alçak Seviyeli Programlama Dilleri

Geliştirilen ilk programlama dilleri, alçak seviyeli programllama dilleri (low-level languages) olarak adlandırılırlar. Bu seviyedeki diller, Makina dili ve Assembly dillerinden oluşmaktadırlar. Makina dili, programlama dillerinin ilkidir. Kullanılabilir ilk dildir. Bütün makina dilleri 0 ve 1 karakterlerinden ibaret olup bu karekterler vasıtasıyla bilgileri işlerler. Her bilgi, iki tabanlı sayı sistemine göre kodlanır. Makina dili düşük seviyeli bir dildir. Çünkü her bilgi bilgisayarın temel özelliklerini dikkate alarak, onlara karşılık gelecek şekilde yazılmalıdır. Makina dilinin bu zorluklarına rağmen, bu dil halen kullanılmaktadır. Yüksek seviyeli dillerle yazılan her program, çalıştırılmadan önce makina diline dönüştürülmelidir. Her tip bilgisayarın, özel, ayrı bir makina dili yapısı vardır. Bu nedenle herhangi bir bilgisayarda makina dili ile gerçekleştirilen bir program, başka bir bilgisayarda çalıştırılamaz.[2]

Assembly dili, Grace Happer tarafından 1950'li yıllarda geliştirilen ve programlama dillerinin ikincisini oluşturan dildir. Assembly dili, makina dilindeki bilgilerin daha kolay yazılmalarına müsade eder. Makina dilinde 0 ve 1 kodlamalarından oluşan bazı işlemler, Assembly dilinde bazı sembollerle

---

[2] Ghezzi, C. ve Jazayeri, M. **Programming Language Conceps.** NewYork, John Wiley & Sons, 1987, s. 198

yapılabilmektedir.[3] Örnek olarak, bir Assembly dilinde, makina dilindekilere karşılık gelen komutlar tablo 1.2'de verilmiştir.

İşlem	Assembly D. Karş.	Makina D. Karş.
Bellekteki değeri register'a ekle	A	01011010
Register ile registeri topla	AR	00011010
Şartlı dallanma	BC	01000111
Bellekdeki değeri karşılaştır	CLC	11010101

Tablo 1.2. Makina Diline Karşılık Gelen Assembly Kodları.

Görüleceği gibi Assembly dili, uygulama programlarını geliştirmeyi ve hatalarını düzeltmeyi makina diline nazaran kolaylaştırır. Gerçekte bir alçak seviyeli dil olan Assembly dili, Makina dilinin sadece bir adım yukarısında görünüyorsa da günümüzde hala geniş bir kullanım alanına sahiptir. Çünkü Assembly dili, makina diline çok yakın olması nedeniyle, yazıldıkları bilgisayarların özel yeteneklerinden faydalanabilir. Pek çok profesyonel program geliştiricileri, kelime işlem, elektronik tablolama ve haberleşme yazılımları gibi programların bazı parçalarını Assembly dili ile yazmayı uygun bulmuşlardır. Bu programlar hem daha hızlı çalışırlar, hem de yüksek seviyeli dillere nazaran, daha güvenli bir şekilde veri depolama ortamlarını kullanırlar. Ancak Assembly dilinde program yazmak, sıkıcı ve zaman alıcıdır. Bilgisayarın donanım ünitelerinde oluşan hızlı teknolojik gelişmelerin getirdiği avantajlarla da artık bu dile olan talep, gün geçtikçe azalmaktadır.

1.1.2. Yüksek Seviyeli Programlama Dilleri

Assembly dili çağını takip eden üçüncü kuşak programlama dilleri, yüksek seviyeli programlama dillerinin (high-level languages) gelişimi ile ortaya çıktı. Bu kategori; COBOL, BASIC, FORTRAN, PASCAL, PL/1 ve ADA dillerini içermektedir. Yüksek seviyeli diller, alçak seviyeli olanlardan, daha az kodlama detayı içermeleri bakımından ayrılırlar. Örnek olarak; yüksek seviyeli bir dilde, 5 farklı sayıyı toplama işlemi X=A+B+C+D+E gibi tek bir program komutu ile ifade edilebilir. Oysa böyle bir işlem, alçak seviyeli bir dilde birkaç komut gerektirir. Bir seferde sadece bir işlem yapabilen basit bir bilgisayar göz önüne alındığında, beş farklı sayıyı toplamak için önce birinci sayı ikinciye eklenmeli, sonra üçüncü sayı ilk toplama eklenmeli ve böyle devam edilmelidir. Makina

[3] Parker, S. ve Case, T. s. 252

dilinde programcı, bilgisayara tüm bu adımları belirtmek zorundadır. Bununla birlikte, yüksek seviyeli diller programcıyı bilgisayarın donanımından uzaklaştırdığı için, düşük seviyeli bir dil kullanılarak makina performansının optimize edilebilmesi yüksek seviyeli bir dil ile mümkün değildir. Bu nedenle, yüksek seviyeli bir dil ile çalışmanın kolaylığının yanısıra, yazılacak programlar makina performansını tam olarak kullanamıyacaklardır.

Üçüncü kuşak programlama dilleri sayesinde programcılık kolaylaşmış, hızlanmış ve pek çok insan için de, artık yabancı bir kavram olmaktan çıkmıştır.[4] Örneğin, bu dillerden biri olan BASIC, kendilerini programcı kabul etmeyen insanlar tarafından bile öğrenilebilir. Ancak daha sonraları geliştirilen yeni kuşak dillerin daha kolay ve çekici olmaları, günümüzde üçüncü kuşak dillere olan ilgiyi de azaltmıştır.

1.1.3. Çok Yüksek Seviyeli Programlama Dilleri

Çok yüksek seviyeli programlama dilleri (very-high-level languages), programcılığı daha kolay hale getirmiştir. Bu diller, dördüncü kuşak diller olarak da adlandırılırlar. Çok yüksek seviyeli bir dilin asıl amacı, kullanıcıların bilgisayara, bir şeyin nasıl yapılacağından ziyade, neyin yapılacağını söylemelerini sağlamaktır.[5] Örneğin, pek çok, yüksek seviyeli programlama dilinde sayıları sıralamak için, 20'den fazla komut ve biraz da ustalık gerekir. Kendi sıralama komutu (SORT) olan dillerde ise, sıralama daha kolay, ancak dosyaları kullanma işi sıkıcı olabilir. Buna karşılık, çok yüksek seviyeli bir dilde ise, kullanıcı sıralanmasını istediği alanı ekranda işaret eder, sonra sıralama işleminin yapılacağı kolon veya sütun işaret edilir, daha sonra da sıralamanın türü belirtilir. Sonra, karmaşık komutlara ihtiyaç duyulmadan sıralama işlemi gerçekleşir. Çok yüksek seviyeli diller, kullanıcıyı, işlemleri gerçekleştirmek için prosedürler tanımlamak mecburiyetinde bırakmadıklarından, bu diller genelde prosedürsüz diller (non-procedural languages) olarak anılırlar. Yüksek seviyeli diller ise prosedürlü dillerdir (procedural languages).

Çok yüksek seviyeli dillerin ilklerinden biri, 1960'ların ortalarında tanıtılan RPG (Report Program Generator) idi. RPG'nin temel felsefesi, programcıların, rapor alabilmek için daha kolay programlar yapabilmeleri

---

[4] Shelly, G. Cashman, T. Waggoner, G. ve Waggoner, W. **Complete Computer Conceps and Microcomputer Applications.** U.S.A., Boyd & Fraser publishing company, 1992, ss. 123-124

[5] Fairley, R. **Software Engineering Concepts.** NewYork, McGraw-Hill, 1993, s. 76

olmuştur. Bir programcı, RPG kullanarak bilgisayar sistemine bir raporun nasıl üretileceğini değil, neye benzemesi gerektiğini bildirir. Böylece programcı, rapora hangi başlıkların koyulacağını, hangi verilerin hangi kolonlara yazılacağını, toplamların ve ara toplamların nereden alınacağı vb. konularda bilgisayara emir vermiş olur. Sonra RPG, bunun en iyi nasıl yapılacağına karar verir ve raporu yazmak için makina dili programını üretir. RPG dili, program yazımında ortaya yeni bir felsefe çıkarmış olsa da, bu konuda eğitim görmeyen kişilerin bu dil ile yazılım geliştirmeleri mümkün görülmemektedir. Bu dil, gelecek olan yeniliklerin habercisi olmuştur. RPG den sonra geliştirilen çok yüksek seviyeli diller rapor üretimi, veri güncelleme gibi işlemleri daha kolay bir hale getirmişlerdir. 1970'lerin ortalarında başlayan mikrobilgisayar sistemleri çağı, dosya yöneticileri, şirket tablolama programları ve veritabanı yönetim sistemleri gibi alanlarda, çok yüksek seviyeli dillerin kullanımını yaygınlaştırmıştır.[6] Bu program paketleri, özel olarak, profesyonel olmayan kişilerin de bilgisayarla ilgilenmelerini ve onların ihtiyaçlarını karşılamayı sağlamıştır. Ayrıca programların daha kısa sürede üretilmelerini sağlamaktadır. Ancak bütün bunlara rağmen, çok yüksek seviyeli diller hatasız değildir, bilgisayar kaynaklarını optimal bir şekilde kullanamazlar ve daha fazla donanım gücü isterler.

Yüksek seviyeli programlama dilleri ile çok yüksek seviyeli programlama dilleri arasındaki farklar tablo 1.3'de gösterilmektedir.

Yüksek seviyeli programlama dilleri	Çok yüksek seviyeli programlama dilleri
Öğrenilmesi ve kullanımı zor	Öğrenilmesi ve kullanımı kolay
Çoğu, profesyonel programcıların kullanımı için geliştirildi	Çoğu, en alt kullanıcılar tarafından bile kullanılabilir
Çoğu, dosya kökenli ortamlarda kullanılmak için tasarlandı	Daha çok veri tabanı ortamlarında kullanılmak için tasarlandı
İşlemler için uzun kodlamalar gerekir	Aynı işi daha az kodlama ile yapar
Program kodlarının okunması ve anlaşılması güç olabilir	Program kodlarının okunması ve anlaşılması kolaydır
Hataları ayıklamak ve bozulmaları önlemek zor olabilir	Kısa ve konuşma diline benzer komutlar ile hata ayıklamak kolaydır
Uygun yapılandırılırlarsa etkin çalışırlar	Aynı iş için daha fazla makina gücünü kullanırlar

Tablo 1.3. Yüksek Seviyeli ve Çok Yüksek Seviyeli Programlama Dilleri Arasındaki Fark Tablosu

---

[6] Pratt, T, W. **Programming Languages:Design and Implementation.** U.S.A., Prentice-Hall, 1984, ss. 65-67

Günümüzde kullanılan bazı alçak, yüksek ve çok yüksek seviyeli dillerden en yaygın olanları ve özellikleri de tablo 1.4'de gösterilmektedir.[7]

Dil adı	Tipi	Tanımı
Ada	Yüksek	Güçlü, çok yapısal
ALGOL	"	Fortran'a benzer bir dil, ama daha esnek
APL	"	Çok özlü, bilimsel dil
BASIC	"	Yeni programcılar için en ideal dil
C	"	Çok yönlü, orta boy bir dil
C++	Çok Yüksek	Popüler nesneye yönelik programlama dili
COBOL	Yüksek	Ticari uygulamalarda en çok kullanılan dil
FORTRAN	"	Bilimsel uygulamalar için en eski yüksek seviyeli dil.
KL1	Çok Yüksek	Paralel programlama dillerinin ilklerinden
OOP PASCAL	"	Popüler nesneye yönelik programlama dili
LISP	Yüksek	Yapay zeka uygulamalarında kullanılan dil
PASCAL	"	Çok kullanışlı, yapısal dil
PL/1	"	Hem ticari, hem bilimsel amaçlı dil
PROLOG	"	Yapay zeka uygulama dillerinden
RPG	"	Rapor üretme dili
Smalltalk	"	İlk nesneye yönelik dil
SQL	"	Yapısal sorgulama dili

Tablo 1.4. Bazı Programlama Dilleri ve Özellikleri

Bilgisayar yazılımı alanında geliştirilen çok yüksek seviyeli dillerin, çok farklı uygulama alanları vardır. Bu diller ile geliştirilen programların, bir kısmı kullanıcılara, bir kısmı programcılara, bir kısmı da her iki gruba hitap etmektedir. Tipik olarak 6 çeşit, çok yüksek seviyeli diller olarak tanımlanan programlama dili vardır. Bunlar sırasıyla; rapor üreticileri, bilgiye erişim ve güncelleme dilleri, karar destek araçları, grafik üreticileri, uygulama paketleri ve uygulama üreticileridir.[8]

---

[7] Shaw, A. "Software Specification Languages Based on Regular Expressions". **in Software Development Tools.** Berlin, Springer-Verlag, 1980, s. 90

[8] Gunther, R. **Management Methodology for Software Product Engineering.** New York, Wiley Interscience, 1978, ss.62-66

1. Rapor Üreticileri: Bir rapor üretici, kullanıcıya hızlı bir şekilde, kendi tasarladığı raporları üretebilmesi imkanını sağlayan bir yazılım ürünüdür. Genelde böyle bir paket, dosyalardan veya veri tabanlarından bilgi çekip, değişik formatlarda raporlar hazırlanmasını sağlar. Yani kullanıcı ve programcılara verileri nasıl şekillendirecekleri ve nasıl görüntüleyecekleri konusunda seçenekler sunarlar. Bir rapor üreticisi programlama dili olan Focus ile kısa bir kodlama yazılarak üretilen örnek bir rapor, tablo 1.5'de verilmiştir.

Focus Kodlaması

```
TABLE FILE SALES
PRINT NAME AND AMOUNT AND DATE
BY REGION BY CITY
IF AMOUNT GT 1000
ON REGION SKIP-LINE
END
```

Elde Edilen Rapor

REGION	CITY	NAME	AMOUNT	DATE
MARMARA	İSTANBUL	ERHAN DEMİR	24.789.000	10/11/96
	BURSA	HASAN YENER	11.000.000	02/12/96
DOĞU ANADOLU	ERZURUM	MEVLÜT KARA	33.660.500	29/10/96
		SELMA AYDENİZ	55.400.000	15/08/96

Tablo 1.5. Bir Focus Kodlaması ve Elde Edilen Rapor

Focus ile rapor üretimi, 6 satırlık kısa bir programla yapılmıştır. Bu kod, SALES adlı dosyada çalıştırılarak, tablo 1.5'deki rapor üretilmektedir. Böyle bir dili, COBOL gibi üçüncü kuşak bir dil ile elde edebilmek için yaklaşık 100 satırlık bir program gereklidir.

2. Bilgiye Erişim ve Güncelleme Dilleri: Bilgiye erişim ve güncelleme işlerini yapmaya imkan veren dillere genelde sorgulama dilleri (query languages) denir. Bilgiye erişim (retrieval) yeteneği, kullanıcıların dosyalardan veya veri tabanlarından bilgi almasını, güncelleme (update) yeteneği ise bilgi ekleme, silme ve değiştirmeyi sağlar. Sorgulama dilleri genelde etkileşimli ortamlarda, belirtilen şartlara uyan verileri bir dosyada veya bir veri tabanında aramakta kullanılır. Örneğin,

```
SELECT SINIF=2, UYRUK='TR'
DISPLAY ALL ÖGRENCİ-NO,AD
```

şeklinde bir sorgulama komutu, bir dosya veya veri tabanından 2. sınıfdaki Türk öğrencilerin numara ve adlarını bulup getirmeye yarar. Geliştirilen bu dillerde, bu tür sorgulamalar sırasında izinsiz erişim ve değişikliklere karşı koruma sağlamak amacıyla, bazı güvenlik önlemleri de alınmıştır.

3. Karar Destek Araçları: Bu araçlar elektronik tabloları, istatistik yazılımları ve modelleme paketlerini içerirler. Bu dil ürünleri kullanıcının karar destek sistemleri oluşturmasını ve bunları bir şeye karar verirken yardımcı olarak kullanmasını sağlar. Kullanıcı, bunları kullanarak kendi veri bankasını oluşturabilir, hesaplamalar yaparak modeller oluşturabilir ve sonuçlar üzerinde analiz yapabilir. Bu paketleri oluşturan çok yüksek seviyeli dillerden en yaygını, LOTUS 1-2-3 programlama dilidir.[9]

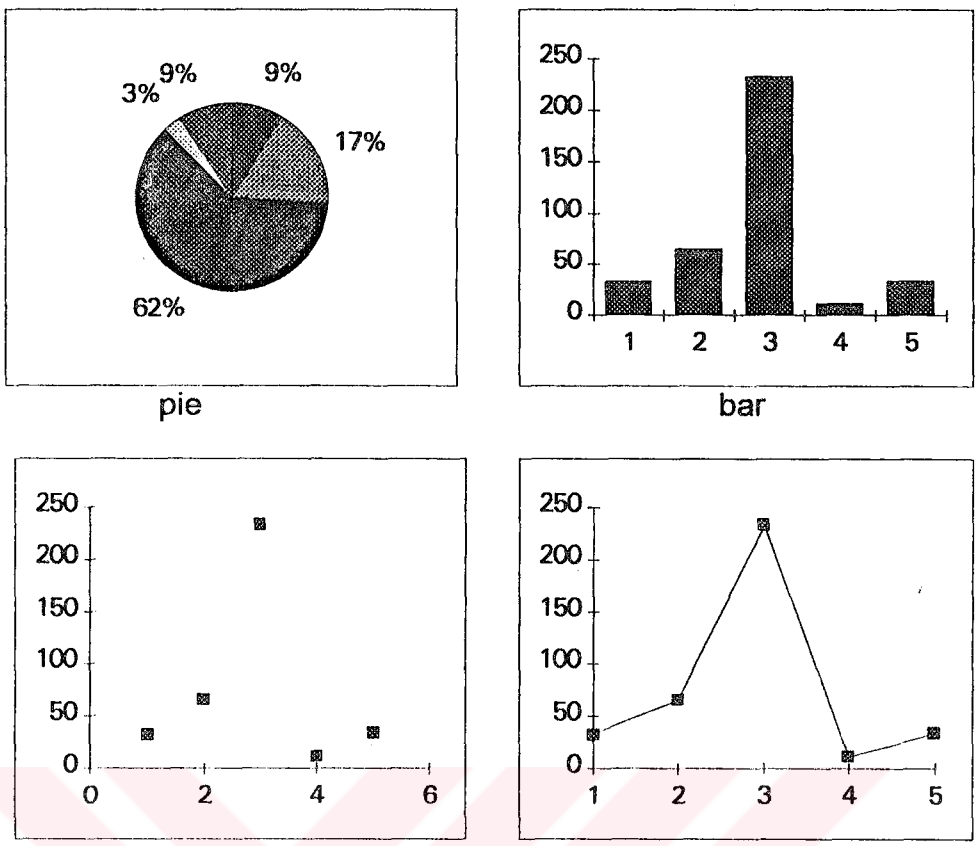
4. Grafik Üreticileri: Grafik sunmayı amaçlayan çok yüksek seviyeli diller, line, bar, pie, scattergram tipinde grafikleri hızlıca üretebilirler. Harvard Graphics ve Freclancd Graphics gibi bazı grafik üreticileri geliştirilmiştir. Bununla birlikte grafik kökenli çok yüksek seviyeli programlama dillerinin çoğu da, elektronik tablolama, modelleme ve raporlama benzeri paketlerin bir elemanı olarak yerleştirilmiştir. Bu programlama dilleri sayesinde, kullanıcının verileri kullanarak istediği şekilde grafikler çizdirmesi çok kolay olmaktadır. Şekil 1.1'de dört farklı türde grafik, örnek olarak verilmiştir.

Bu grafikler, karar vermeye yardım eder ve uygulamalarda önemli bir rol oynar. Olaylara profesyonel bir bakış getirir.

5. Uygulama Paketleri: Uygulama paketleri, faturalama, bordro hazırlama, kelime işleme, müşteri hesabı tutma gibi ticari uygulamalar için tasarlanmış programlar olarak bilinirler. Bunların ticari pazarda mainframe, mini ve micro bilgisayarlarda çalışabilen versiyonları vardır. Basit olarak, böyle bir paketi satın alan kişiler, kendilerine ait özel parametrelerini girdikten sonra program paketini çalıştırabilirler. Farklı kullanıcılar için konfigürasyon ayarı yapılabilir. Geçmişte bu tür paketler, esnek değildiler ve farklı kullanıcıların ihtiyaçlarını görmezlerdi. Genelde isteyen kullanıcı için, neredeyse özel olarak hazırlanırlardı. Ancak, son yıllardaki gelişmelerle, bu tür uygulama programları esnekleştirilmiş ve benzer konularda çalışan kullanıcıların isteklerini, parametreleri ayarlanmak suretiyle karşılar olmuştur.

---

[9] Gunther, R. s. 67



Şekil 1.1. Çok Yüksek Seviyeli Programlama Dilleri İle Oluşturulan Grafikler

6. Uygulama Üreticileri: Bir uygulama üretici, kullanıcı veya programcıya, bir uygulamanın içerdiği tüm program kümesini hızlıca oluşturma imkanı veren çok gelişmiş dördüncü kuşak bir programlama dilidir. Bu araçlarla yapılan çalışmalarda, kullanıcı nelerin yapılması gerektiğini belirtir, uygulama üretici ise bu işe uygun program kodlarını üretir. Bazıları giriş, güncelleme ve işleme için kod üretirken, bazı uygulama üreticileri ise, etkileşimli (on-line) olarak, kullanıcının terminalin karşısına oturup girdiği verileri, işlemleri, rapor formları vb. bilgileri cevaplandırmasına imkan verir.[10]

1.1.4. Dördüncü Kuşak Sonrası Diller

Dördüncü kuşaktan sonra geliştirilen programlama dillerinde, oldukça fazla değişiklik olmuştur. Bu dönemdeki programlama dilleri, dördüncü kuşak sonrası diller (beyond the fourth generation) veya beşinci kuşak programlama dilleri olarak adlandırılırlar. Dördüncü kuşak sonrası diller, dört önemli alanda

[10] Gunther, R. s. 68

gelişme göstermiştir. Bunlar; uzman sistemler (expert systems), doğal diller (natural languages), nesneye yönelik diller (object-oriented languages) ve paralel programlama dilleridir.[11]

Bu programlama tekniği ile, belirlenen alanlarda hedeflenen potansiyel verimlilik, henüz tam anlamıyla gerçek verimliliğe dönüşmemiştir. Yüksek donanım gücü isteyen bu tekniğin arzulanan hedeflere ulaştırılması, günümüz bilgisayarcılarının çalışma konularını oluşturmaktadır.

1. Uzman Sistemler: Uzman sistemler, uzman kişilerin normal niteliklerini kıyaslayan ve zekalarını örnek alan yazılım mantığıdır. Bu sistemlerin amaçları, insan ile makina arasında arayüzlerin geliştirilmelerini sağlamaktır. Bunun yanı sıra, bu gibi yazılımlar bilgisayar sistemlerinde, programlardaki mantık hatalarını ve veri girişindeki insan hatalarını elimine edebilirler. Aynı zamanda, birçok kritik tasarım alanlarında insanlardan alınan verilerin değerlendirilmesi sonucunda, bilgisayar sistemlerinin geliştirilmesine neden olabilir.[12]

2. Doğal Diller: Programlama çalışmalarının en önemli hedeflerinden birisi, kullanıcı arayüzlerinin oluşturulmasıdır. Sıradan programlamalarda, program geliştirmek isteyen kişiler, öncelikle bir programlama dilini öğrenmek mecburiyetindedirler. Böyle bir mecburiyet olmasa, gerek programcılar, gerekse kullanıcılar bilgisayarla daha çok uğraşma ve daha fazla yazılım üretme şansına sahip olacaklardır. Artık insanlar, bilgisayarlarla irtibat kurmayı, kendi ana dilleriyle yapmayı, işlemleri normal konuşma veya yazışma dilleri ile gerçekleştirmeyi arzu etmektedirler. Bu durum, özel bir programlama dilinin içeriğini veya kurallarını öğrenmeyi, kullanıcılar veya programcılar için gereksiz kılmaktadır. Bunun yerine bilgisayarlarla ilgili istekler, doğal dil sistemi ile bilgisayara aktarılacak ve makina diline dönüştürülecektir. Doğal dil uygulama sistemleri, hala yetersizdir. Ancak bu konuda da hızlı bir gelişim gözlenmektedir.

3. Nesneye Yönelik Diller: Nesneye yönelik diller, nesneleri ve mesajları kullanırlar.[13] Nesneler, işlemlerin gerçekleştirileceği soyut veya somut bilgilerdir. Mesajlar ise, nesneler arasındaki iletişimlerdir. Nesneye yönelik

---

[11] Parker, S. ve Case, T. ss. 255-260

[12] Hunt, V, Daniel. **Artificial Intelligence & Expert Systems Sourcebook.** Newyork, Chapman & Hall, 1986, s. 52

[13] Booch, G. "Object-Oriented Development". **IEEE Transactions on Software Engineering.** U.S.A., IEEE, 1986, s. 96

dillerde, ne yapılacağı ve nasıl yapılacağı konularında nesnelere mesajlar verilmektedir. Örneğin, ekran üzerindeki bir kürsor nesne olabilir. Ona ait bir mesaj, bir çizgi çizerek diğer pozisyona geçişin sağlanması işlemidir. Nesneye yönelik diller, özellikle grafik tabanlı kullanıcı arayüzleri için çok uygundur. Nesneye yönelik programlama dillerinin üstünlüklerinden birkaçı, program geliştirme zamanının kısa olması, programlamada özelliklerin hep nesne üzerinde toplanması nedeniyle programlama için gerekli olan kodlamaların az olması ve kodun yeniden kullanılabilirliğinin yaygın olarak mümkün olmasıdır.[14]

Nesneye yönelik yaklaşımın ilk orijinal örneği, Smalltalk programlama dilidir. Sonraları C ve Pascal dillerinin son versiyonları olan, C++ ve OOP Pascal da nesneye yönelik yaklaşımı temsil eden, dördüncü nesil sonrası programlama dillerini oluşturmuşlardır. Nesneye yönelik dillerin, yüksek verimlilik potansiyeline sahip oldukları ve genellikle endüstri alanında en çok kullanılan bu dillerin, üretilen kodun yeniden kullanımına imkan sağlayan diller olduğu, bir çok programcı tarafından geliştirilen uygulamalar aracılığıyla görülmüştür.

4. Paralel Programlama Dilleri: Paralel işlem, aynı anda çalışan birkaç merkezi işlem üniteleri üzerinde çalışabilmek için iş bölümünü gösterir.[15] Donanım fiyatlarının eskiye nazaran nispi olarak azalması ile birlikte bu yaklaşım, bilgi işlem sürecini hızlandıran en pratik yöntemdir. Ayrıca bu tür programlama dilleri, bilgisayar programlarının kodlama şekillerini kendilerine has bir şekilde değiştirirler. Günümüzde çok az sayıda paralel programlama dili mevcuttur. Yeni nesil bilgisayar teknolojisi için Japon endüstrisi tarafından geliştirilen, KL1 en yaygın kullanılan paralel programlama dilidir.

1.2. YAZILIM GELİŞTİRME SİSTEMLERİ

Her alanda kullanabileceğimiz, en önemli temel öğelerden birisi, bilgidir. Bilgi, kısaca işlenmiş veri olarak tarif edilebilir. Geliştirilecek bir sistemde, önce sahip olunması gereken bilgiler temin edilmelidir. Bu bilgilerin elde edilmesi hem aşamalar halinde, hem de farklı yöntemler ile olabilmektedir. Farklı aşamalarda ve farklı yöntemlerle elde edilebilen bilgiler ile, yeni bir sistem

---

[14] Florentin, J, J. **Object Oriented Programming Systems Tools and Applications.** Newyork, Chapman & Hall, 1991, s.114

[15] McCorduck, P. ve Feignebaum, E. **The Fifth Generation.** New York, Addison-Wesley, 1983, s. 87

geliştirilebilmesi mümkün olur. Sistem kavramı, günümüzde hemen her bilim alanına girmiş ve düşünme biçimini etkilemiştir. Sistem kavramını esas alan düşünce biçimi, endüstriyel işletmelerden, en saf bilim alanlarına kadar hemen her alanda büyük bir öneme sahiptir. Her zaman plan, yöntem, sıralama ve düzenleme kavramlarını hatırlatan sistem sözcüğü, bilim adamlarının ve araştırmacıların amaçlarına ulaşmak için çok kullandıkları bir kavramdır. Genel kabul görmüş tanımını şu şekilde verebiliriz: Sistem, bir veya daha çok amaca veya sonuca ulaşmak üzere aralarında ilişkiler olan fiziksel ve kavramsal, birden çok bileşenin oluşturduğu bütündür.[16] Bu amaç doğrultusunda, bir bilişim sisteminin oluşturulabilmesi için, sistemin geliştirilme aşamaları ve yöntemleri incelenmelidir.

1.2.1. Bir Bilişim Sisteminin Geliştirilme Aşamaları

Yeni bir bilişim sisteminin geliştirilmesinde, planlama oldukça önemli bir yere sahiptir. Bundan sonra gerçekleştirilecek tüm faaliyetlerin başarısı, büyük ölçüde bu planlamanın yapılmasına bağlıdır.[17] Planlama, herhangi bir sistem oluşturulurken, işlemlerin nasıl ve hangi aşamalarda yapılacağının, en başta tesbit edilmesidir.[18] Planın geliştirilmesi sırasında standart bir terminolojiye dikkat edilmelidir. Amaçlar, çalışmanın kapsamı, sınırlamalar, çalışmanın aşamaları, tamamlanma tarihi, tahmini maliyet, çalışmayı yürütecek ekip gibi konular sistem geliştirilmesi sırasında planlanması gereken hususlardır.

Bir bilişim sisteminin geliştirilmesinde, planlama esas alınarak önce mevcut sistemin analizi yapılmalı, sistemin yetersiz olması, verimliliğinin azalması veya teknolojisinin eskimesi gibi nedenlerle yeni sistemin geliştirilmesine karar verilmesi halinde, ikinci adım olarak yeni sistemin tasarımı yapılmalı ve test edilmelidir. Test aşamasının olumlu sonuçlanması halinde ise, uygulama aşamasına geçilmelidir.

1.2.1.1. Mevcut Sistemin Analizi

Yeni bir sisteme ihtiyaç duyulmasının en önemli nedenlerinden biri, eski sistemin yeni uygulamalara veya gerekli değişikliklere uyma yeteneğinin, ve

---

[16] Esen,H,Öner. **İşletme Yönetiminde Sistem Yaklaşımı.** İstanbul, Bayrak Matbaacılık, 1985, s.1

[17] Esen, s.72

[18] Locus, H. C. **The Analysis, Design, and Implementation of Informations Systems.** New York, McGraw-Hill, 1992, s. 33

büyüğebilme imkanlarının kalmamış olmasıdır. Planlanmış ihtiyaçların gerçekleşmesi arzusu, sürekli dinamik bir yapıya ihtiyaç göstermektedir. Ayrıca, sistemin artık kontrol edilememesi veya herhangi bir müdahalenin büyük zaman, emek ve masraf gerektirmesi durumlarında, yeni bir sisteme ihtiyaç duyulabilir. Bu aşamada bütün bunlar dikkate alınarak, mevcut bilgi sisteminin amaçları, sınırları, araçları, etkinliği, bileşenleri ve fonksiyonel ilişkileri incelenerek yetersiz veya gereksiz olduğu kısımlar belirlenmelidir.

Bunun için, öncelikle mevcut sistem hakkında bilgi toplanmalı, yazılı kaynaklar, kodlamalar, şemalar incelenmeli, sistemi uygulayan kişilerle temasa geçilerek aksaklıklar en iyi kaynaktan öğrenilmelidir. Bunlara ilaveten, yöneticilerin ve uygulayıcıların güncel istek ve arzuları, mevcut sistem ile karşılaştırılmalıdır. Faaliyetlere ilişkin bilgi sisteminin akış şemaları, sistem tasarımı ile ilgili organizasyon ve karar tabloları hazırlanmalıdır. Hazırlanan bu belgeler sistem analisti tarafından sistemin verimliliği, maliyet, zaman ve kalite bakımından değerlendirilerek yeni bir bilişim sisteminin oluşturulup oluşturulmayacağına karar verilmelidir. Analist, elde edilen belgeler ve bilgiler ışığında, öncelikle mevcut sistemin amaçlarının ve belirlenmiş kısıtlamaların hangi ölçülerde gerçekleştiğini belirlemeli, işlemlerin kimler tarafından ve nasıl yürütüldüğünü gözlemlemeli, mevcut sistemin performansının artırılıp artırılamayacağını araştırmalı, sistemi, maliyet, zaman, kalite ve etkinlik bakımlarından değerlendirmelidir.[19]

Mevcut sistemin analiz işlemleri tamamlandıktan sonra, sistem analisti tarafından bir rapor düzenlenerek, çalışmalarda elde edilen belge ve bilgiler ışığında alınan kararlar, bu raporda belirtilmelidir. Bu raporun amacı, hem yapılan çalışmalar hakkında gerekli yerlere bilgi vermek, hem de bir sonraki tasarım aşamasına kaynak oluşturmaktır. Raporda genellikle şu hususlara yer verilir:

1. Sistem analizinin amaçları ve kapsamı,
2. Mevcut sistemin ve onu oluşturan süreçlerin problemleri,
3. Analiz işlemi sırasında ortaya çıkan sınırlamalar.
4. Bu aşamada çare olabilecek çeşitli seçenekler,
5. Mevcut sistemi yeniden düzenlemek veya tasarlanan yeni sistemi gerçekleştirmek için gerekli tüm ihtiyaçların tahmini.

Bu işlemlerden sonra, tasarım ve test aşamalarına geçilebilir.

---

[19] Jackson, M. **System Development.** Englewood Cliffs, New Jersey, 1983, ss. 71-72

1.2.1.2. Yeni Sistemin Tasarımı ve Test Edilmesi

Yeni sistemin tasarımı ve test edilmesi, mevcut sistemin tanınması ve analizini izleyen ve üretkenlik isteyen bir faaliyettir.[20] Analiz aşamasında eksiklikler ortaya çıktıkça, analist tarafından bunların giderilmesi için yeni sistem oluşturulmaya başlanır. Mevcut sistemin olmadığı durumlarda analiz ve tasarım aşaması tek kabul edilerek, muhtemel eksikliklerin olmaması için çözüm yolları geliştirilir. Kaldı ki mevcut sistemin bulunmadığı zamanlarda bile, analiz ve tasarım aşamaları arasında teoride olduğu gibi bir ayrım gerçek uygulamalarda bulunmamaktadır.

Tasarımın amaçlarından birisi, basitliktir.[21] İleride ihtiyaç duyulabilecek iyileşme, genişleme ve taşınabilirlik gibi bazı durumların oluşumuna açık bir sistem geliştirilmelidir. Sistem tasarlanırken, ileride olabilecek bir dizi değişikliğe rağmen, hala, basit görünümünü koruyabilecek şekilde planlanmalıdır. Tasarım esnasında sadece bir programın oluşturulup çalıştırılması değil, projenin bir bütün olarak ele alınması ve daha sonra gerçekleştirilecek değişiklikler vs. için gerekli önlemlerin düşünülmesi gerekir.

Bir sistem tasarlanırken çeşitli aşamalardan geçilebilir. Bu iş için uygun aşamalar şunlardır:

1.Yeni bilişim sisteminin amaçlarının belirlenmesi
2. Bilgi işleme sürecinin sınırlarının belirlenmesi
3. Kullanılacak donanım kapasitelerinin belirlenmesi
4. Maliyet / Fayda analizinin yapılması
5. En uygun olduğu düşünülen tekniğe ait adaptasyon planının oluşturulması

Tasarımın son aşamasını test oluşturur. Test, ileride telafisi mümkün olmayan zaman ve emek kaybına yol açmamak için, uygulama gerçekleştirilmeden önce yapılmalıdır. Testi yapılmamış bir yazılım, aslında uygulanamaz demektir.

---

20 Peterson, J. L. ve Silbershatz, A. **Operating System Concepts.** New York, Addison-Wesley, 1985, s. 87
21 Jackson, M. s.75

İyi bir tasarım yapabilmek için, uygulamaya yönelik bazı tavsiyeler şu şekilde sıralanabilir:

1. Yapılmak istenenin ne olduğu tam olarak belirlenmeli
2. Amaçların açık ve somut olmasına dikkat edilmeli
3. Var olan sistemlerden esinlenerek, başlangıç noktası oluşturacak bir model kullanılmalı
4. Esneklik, geliştirme, taşınma ve tekrar kullanım özellikleri dikkate alınmalı
5. Kavramları belirtmek üzere sınıflar kullanılmalı
6. Kütüphanelerin ve sınıfların tekrar kullanımı için dokümantasyona önem verilmeli
7. Geliştirilen bir arabirimin diğer bir arabirime bağlılığı en aza indirilmeli
8. Herşey mümkün olduğu kadar (aşırılığa kaçmamak şartıyla) basit halde ve küçük boyutta tutulmalı
9. Etkinlik gözardı edilmemeli
10. Test işlemi yapılmadan uygulamaya geçilmemeli
11. Tasarımcıların, programcıların ve kullanıcıların birer insan oldukları unutulmadan ve kullanıcıların herzaman daha az bilgi ve dikkate sahip olacakları göz önünde bulundurularak, geliştirilecek arabirimlerin uygunluğuna ve sistemin sağlamlığına dikkat edilmelidir.

1.2.1.3. Yeni Bilişim Sisteminin Adaptasyonu

Yeni sistem tamamen test edildikten sonra, eski sistemden yeni sisteme geçilebilir. Bu geçiş sonunda, sadece bilgi işleme düzeninde değil, kullanılan donanımda, teçhizatta ve hatta personelde değişiklik sözkonusu olabilir. Yeni sistemin uygulamaya geçirilmesi 4 farklı biçimde gerçekleştirilebilir.[22]

1. Paralel Geçiş: Bu yaklaşımda, belirli bir süre eski sistemle yeni sistem beraber işletilir. Bu yaklaşımın en büyük üstünlüğü, yeni sistemde başlangıçta çıkabilecek aksaklıklardan organizasyonun etkilenmemesi ve yeni sisteme karşı bir tepkinin doğmamasıdır. Yine bir diğer üstünlüğü ise, kullanıcının her iki sistemi karşılaştırma imkanına sahip olmasıdır. En büyük sakıncası ise, iki

---

22 Gremillion, Lee. ve Philip, J. Pyburn. "Justifying Decision Support Systems and Office Administration Systems". **Journal of Management Information Systems.** New York, Addison-Wesley, 1985, s. 87

sisteme aynı anda tahsis edilecek kaynak ve sermayenin diğer geçiş şekillerine göre fazla olmasıdır.

2. Pilot Sistem Aracılığı İle Geçiş: Bu yaklaşımda, yeni sistem organizasyonun sadece küçük bir bölümünde uygulanır. Bu geçiş, beklenmeyen büyük problemlerin çıkması halinde faaliyetlerin durdurulmasını önler. Bu şeklin en büyük sakıncası ise, yeni sisteme tam geçişi geciktirmesidir.

3. Aşama Aşama Geçiş: Bu yöntemde, eski sistem, önceden belirlenen bir süre içersisinde yeni sistemle yavaş yavaş değiştirilir. Bölümlere ayrılan yeni sistem, aşama aşama eski sistemle yer değiştirerek organizasyonda yerini alır. Bu sistemin sakıncası ise, yer değiştirmeler sırasında çıkabilecek koordinasyon sorunudur. Bu sorun, yeni sistem hakim olmaya başladıkça azalır ve kaybolur.

4. Doğrudan Geçiş: Bu geçiş şeklinde yeni sistem bütün fonksiyonları ile birlikte toplu olarak uygulamaya konulur. Geçiş sonrası yeni sistemde problemlerin olması halinde, eski sisteme yeniden dönme şansı yoktur. Bu nedenle risklidir. Ancak eski sistem hiç görev yapamıyorsa veya yeni sistem eski sistemden çok farklı ise, bu geçiş şekli uygun olur. Diğer hallerde geçiş, yüksek risk taşıdığından, bu geçiş şekli kullanılmadan önce çok dikkatli olunmalı, tüm programlar ve sistem iyice test edilmiş olmalıdır.

1.2.2. Bir Bilişim Sisteminin Geliştirilme Yöntemleri

Sistem geliştirme, günümüze kadar klasik, genelleştirilmiş ve en son olarak da nesneye dayalı olarak adlandırılan üç farklı yöntemle sağlanmıştır.[23]

1.2.2.1. Klasik Yöntem

En eski yazılım geliştirme yöntemidir. Geliştirildiği dönemlerde ihtiyaca cevap vermiş olmasına rağmen, gerek yazılım, gerekse donanım özelliklerinin gelişmesi, bu yöntemi yetersiz kılmıştır. Belirlenen amaçlar ve verilen sınırlamalar dikkate alındığında, çok fazla esnek olmayan bir yöntem olduğu görülür. Esnekliğin olmaması, geliştirilen sistemlerin ileride karşılaşılabilecek

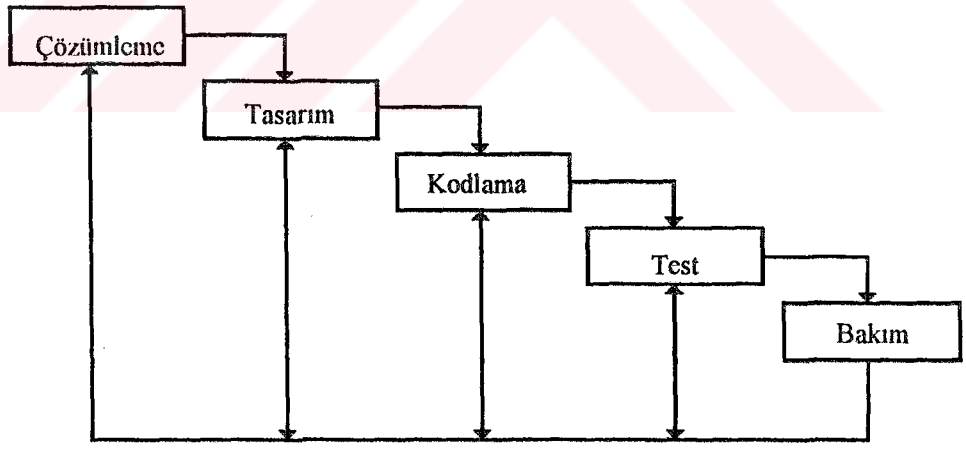
---

[23] Locus, H. C. ss. 45-49

değişikliklere cevap verilebilmesini engeller. Bu yöntemde proje gelişimi, şu aşamalardan oluşmaktadır:

1. Çözümleme
2. Tasarım
3. Kodlama
4. Test
5. Bakım

Çözümleme aşamasında konu incelenir, analiz edilir, ihtiyaçlar tesbit edilir ve bütün bu hususlar proje tasarımında kullanılmak amacıyla, herhangi bir şekilde belgelendirilir. Tasarım aşamasında, var olan belgeler incelenerek proje baştan sona adım adım tasarlanır. Kodlama aşamasında tasarım planı dikkate alınarak, gerçekleştirilecek projenin kodlama işlemi yapılır. Yazılım, genellikle tek bir programlama dili kullanılarak, bir ana veya alt programdan veyahut en çok birkaç alt program ile ana programdan oluşur. Test aşaması ise, yazılımın, ihtiyaçları karşılayıp karşılamadığı, çalışma anında hataların oluşup oluşmadığı gibi konuların araştırıldığı aşamadır. Bakım işlemi de, yazılım geliştirme aşamalarının dışında gibi görünse de, aslında proje geliştirme işleminin son aşamasını oluşturur.[24]



Şekil 1.2. Klasik Yöntem

Klasik yöntemde, şekil 1.2'de gösterildiği gibi geliştirilecek sistemin önce bir çözümleyici aracılığıyla, çözümlenmesi ve analizi yapılır. Sonra tasarlanır ve

[24] Güven,Doğan. **Bilişim Dizgilerinin Çözümleme Tasarım ve Gerçekleştirimi için Görsel Nesneye Dayalı bir CASE ortamı.** İzmir, Basılmamış Doktora Tezi, 1991, s. 14

belirlenen programlama dili ile kodlaması yapılır. Daha sonra ise test işlemi gerçekleştirilir. Testin başarılı olması halinde, geliştirilen projenin uygulanması mümkün olur. Klasik yöntemde, uygulama aşaması, aslında sürekli bakım gerektiren bir aşama olarak kabul edilir.[25] Bakım aşamasında, projenin deforme olmaması için gerekli düzenlemeler yapılır.

Sistem geliştirilmesinde, klasik yöntemin uygulanması ile karşılaşılan bazı sakıncalar şunlardır:[26]

1. Günümüzde, gerçekleştirilen ve çeşitli alanlarda kullanılan projeler uzun süre değişmeden kullanılamaz. Bu projelerin sık sık yenilenmesine ihtiyaç vardır. Bu durum klasik yaklaşımda mümkün değildir.

2. Uzun ömürlü olması istenilen bir proje için geliştirilecek yazılımda ne kadar istenirse istensin bütün özellikler belirtilemez. Bu nedenle noksan kalacak özelliklerin projeye adaptasyonu bu yöntemle mümkün değildir.

3. Geliştirilen bir yazılımın hataları, uygulamaya geçilmeden tam manasıyla görülemez. Klasik yöntemde ise uygulamaya girmiş olan bir proje üzerinde değişiklik yapabilmek pek mümkün olmaz.

4. Bu yaklaşımla gerçekleştirilen projelerde, bakım, onarım işlemleri çok zordur. En küçük bir hatanın giderilmesi, bütün kodlamayı etkiler.

5. Bu projelerin, başka projeler tarafından kullanılması veya desteklenmesi söz konusu değildir.

1.2.2.2. Genelleştirilmiş Yöntem

Bu yöntem klasik yöntemin daha pratik bir şekli olarak oluşmuştur. Genelleştirilmiş yöntem, yenilenme, hata düzeltme ve yeni ilaveler yapılabilmesi konularında klasik yöntem kadar yetersiz olmasa da istenilen seviyede değildir. Klasik yöntemden farkı, aşamaların birbirleri içerisinde kullanılabilir olmasıdır. Genelleştirilmiş yöntem, klasik yöntemin çözümleme ve

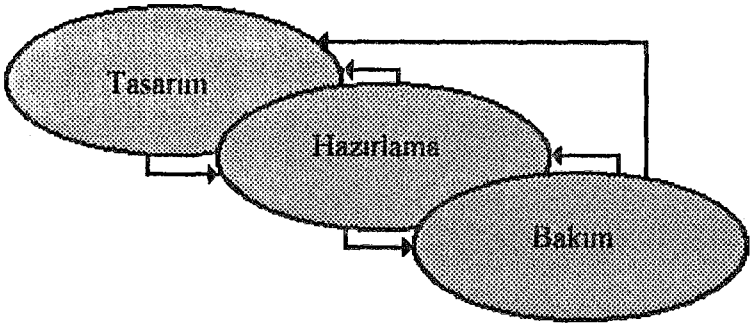
---

[25] Millman, Howard. "There's Something for All in Applications Generators".**Computerworld.** New York, McGraw-Hill, 1992, ss.45-46

[26] Filman, R. ve Friedman, D. P. **Coordinated Computing: Tools and Techniques for Distributed Software.** New York, McGraw-Hill, 1984, s.63

tasarım aşamalarını tek aşama halinde tasarım adı ile tanımlamıştır. İkinci olarak klasik yöntemdeki kodlama ve test aşamaları, bu yöntemde hazırlama aşaması olarak alınmaktadır. Son aşama ise, yine bakım aşamasıdır. Aslında pratikte çözümleme ve tasarım aşamaları iç içedir.[27] Bunların ayrı düşünülmeleri teorik olarak mümkün olsa da, pratikte çok güçtür. Kodlama ve test aşamaları da yine birbirlerinden çok fazla ayrılamayan aşamalardır.

Bu yöntem şekil 1.3'de gösterilmektedir.



Şekil 1.3. Genelleştirilmiş Yöntem

Tasarım aşamasında konu incelenir, ihtiyaçlar ve kaynaklar tesbit edilir. Yapılan tesbitler bir rapor halinde düzenlenerek yazılım için hazırlanır.

Hazırlama aşamasında, tasarlanan projenin yazılımı gerçekleştirilir. Kullanılacak programlama dili, veri yapısının tesbiti ve yazılımın kodlanmasından sonra test işlemlerinin de yapılması ile bu aşama sonuçlandırılır.

Genelleştirilmiş yöntemin son aşamasını ise, bakım oluşturur. Bu aşamada, istenilen seviyede olmamakla birlikte hata düzeltme, kullanıcıların yeni isteklerinin projeye eklenmesi ve gerekli düzenlemeler yapılabilir.

1.2.2.3. Nesneye Yönelik Yöntem

En son yazılım geliştirme yöntemi olan nesneye yönelik yöntem, çok daha karmaşık projelerin kolayca düzenlenebilmesini, bölümler halinde değiştirilip geliştirilebilmesini sağlamayı amaçlar.[28]

---

[27] Fairley, R. "A Model of Software Structure". **Proc. 17th Hawaii Intl. Conf. on System Sciences.** Western Periodicals, 1984, s. 192

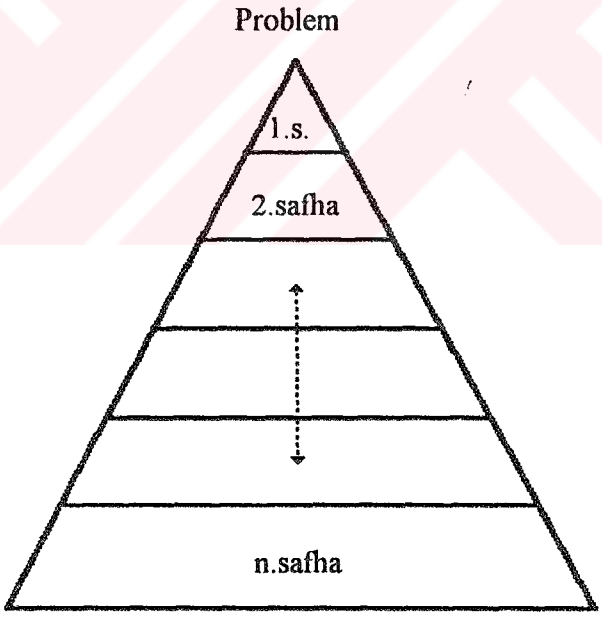
[28] Güven, s. 16

Geliştirilme veya değiştirilme durumlarında proje, bir bütün olarak değil de, parçalar halinde ele alınır. Bu yöntemin temel dayanağı, adını da taşıdığı nesnedir. Nesneyi en basit anlamayla bir varlık olarak düşünebiliriz. Yani yazılımda esas alacağımız nesneler, gerçek dünyadaki soyut ve somut varlıklar gibi davranış gösteren program parçaları ve verilerdir. Verilerin nesne anlamında kullanılabilmesi için, verinin kendisi ile o veri üzerinde tanımlanan işlemlerin bir bütün halinde düşünülmesi gerekir.

Nesneye dayalı yöntemin bazı temel prensipleri şu şekildedir:

1. Her nesnenin onu temsil eden bir özelliği söz konusudur.
2. Bu özellik dış etkenler ile değiştirilebilir.
3. Bir nesnenin bütün özelliklerinin oluşturduğu küme o nesnenin davranışını belirler.
4. Aynı özelliklere sahip nesneler nesne sınıflarını meydana getirirler.
5. Bu sınıf özelliği sayesinde işlemlerin tek tek nesneler üzerinde yapılması yerine nesne grupları üzerinde yapılması sağlanır.[29]

Nesneye yönelik yöntemin gelişimi, şekil 1.4'de gösterilmektedir.



Şekil 1.4. Nesneye Yönelik Yöntem

[29] Florentin,J,J. **Object Oriented Programming Systems Tools and Applications.** Newyork, Chapman & Hall, 1991, s. 65

Nesneye yönelik yöntemde, problemin başlangıcından çözüme doğru safhaların sıralar halinde takip edilmesi esastır.[30] Yani her safha kendi içerisinde tamamlanır ve bir daha ele alınmaz. Gelecek safhalarda tekrar işlenmez. Her safhaya ait işlemler ayrı ayrı tamamlanır ve test edilebilir. Şekil 1.4'de de görüldüğü gibi, projenin gelişim evresinde her safhaya ait fonksiyonlar kendi içerisinde tamamlanır, ancak sistemin bir bütün haline gelebilmesi bütün safhaların tamamlanarak projenin sonuna gelinmesiyle gerçekleşir. Sistem için gerekli bazı özel durumlar, sistemin tamamlanmasına kadar her safhada hazırlanabilir. Yani planlama proje boyunca devam etmektedir.

[30] Robinson, Peter. **Object Oriented Design.** Newyork, Chapman & Hall, 1992, s. 205

İkinci Bölüm

NESNEYE YÖNELİM

Günümüzde gelişen bilgisayar teknolojisi, yeni bilgisayar kullanım tekniklerini de beraberinde getirmektedir. Bu tekniklerden bir tanesi ve en güncel olanı, nesneye yönelik yaklaşım tekniğidir. Bu yaklaşımda temel amaç, gerçek hayattaki nesnelerin hareketlerini örnek alarak ve nesneleri özelliklerine göre sınıflandırmak suretiyle kompleks problemlerin çözümünü gerçekleştirmektir.[31]

Bu tekniğin uygulanabilmesi için, nesne kökenli programlama dilleri geliştirilmiştir. Bu diller yapıları gereği "obje" oluşumuna müsade ederler. Obje oluşumunda nesne ve özellikleri, aynı anda tanımlanma imkanına kavuşurlar. Gerçek hayattaki olayları, doğrudan doğruya programdaki objelere yerleştirmek mümkündür. Nesneye yönelik programlama dilleri, nesneler üzerindeki genel ilkelerin, program yazımında kullanımına izin vermekle beraber nesneler arasındaki farkların da belirlenmesini sağlar.

2.1. NESNEYE YÖNELİMİN GELİŞİMİ

Nesneye yönelimin tarihçesi, Smalltalk programlama dilinin tarihçesi ile özdeşleştirilebilir. Çünkü, bugün nesneye yönelimde kullandığımız kavramların hemen hemen hepsinin oluşumu ve gelişimi, Smaltalk'un gelişimine paralel ve bağımlı olmuştur.[32]

Nesne kavramı, ilk defa 1967 yılında gündeme gelmiştir. Oslo'daki Norveç Bilim merkezinde çalışan Ole-Johan Dahl ve Kristan Nygaard tarafından, bir benzeşim dili olarak geliştirilen Simula-67, nesne kavramını ilk defa içeren dildir.[33] Ancak bu düşünce, yeterince olgunlaşmadığı için olsa gerek o günlerde pek ilgi görmedi.

---

[31] Blair, G. Gallagher, J. Hutchisaon, D. ve Shepherd, D. **Object Oriented Languages Systems and Applications**. Newyork, Halsted Press, 1988, s. 78

[32] Goldberg, A. ve Rabson, D. **Smalltalk-80 The Language**. USA, Wesley Series in Computer Science, 1989, s. 66

[33] Goldberg, A. **Smalltalk-80 The Interactive Programming Environment.** USA, Wesley Series in Computer Science, 1984, s. 120

1970'lerin başında, Xerox'a bağlı Palo-Alto Araştırma Merkezinde (PARC), Öğrenmeyi Araştırma Grubu (Learning Research Group), insanların bilgisayarları daha istekli kullanabilmelerini amaçlayan bir çalışmaya başladı.

1972'de, öğrenmeyi araştırma grubu'nda görevli Alan Kay, Adele Goldberg ve arkadaşları, tümüyle nesneye yönelik bir dil olan Smalltalk-72 uyarlamasını geliştirdiler. Bu çalışma, yeterli düzeyde olmadığı için dış dünyaya açılamadı.

1976'da Smalltalk-76 versiyonu geliştirildi. Eski uyarlamasında bir ASCII terminalini andıran kullanıcı arabirimi geliştirilerek, nokta-grafik tabanlı ve mause ile yönetilen bir kullanıcı arabirimine dönüştürüldü.[34]

Mouse ile yönetilen nokta-grafik tabanlı kullanıcı arabirimi düşüncesi, ilk olarak Apple tarafından üretilen Lisa adlı bilgisayarda boy gösterdi, daha sonra da neredeyse Macintosh bilgisayarlar ile bütünleşti.

1980 yılında Smalltalk son halini aldı ve Smalltalk-80 versiyonu üretildi. Bu uyarlama daha taşınabilir özelliktedir. Bundan önceki Smalltalk uyarlamaları, sadece Xerox üretimi Dorado, Alto ve Dolphin adlı bilgisayarlarda çalışıyordu. 80 uyarlaması, aynı zamanda ilk Smalltalk standardı olarak da kabul edilebilir.

Ağustos 1980'de piyasada bulunan donanımların, Smalltalk için yeterli olmadığının bilincinde olan Xerox, üretilen yazılıma uygun donanıma sahip bilgisayarların üretilmesi ve yaygınlaşması amacıyla Apple, Digital Equipment Corporation, Hewlett-Packard ve Tektronix'ten gözlemciler çağırarak Smalltalk sistemini tanıttı.

Bir yıl sonra 1981 yılı sonlarına doğru ünlü bilgisayar dergisi Byte'ta yer alan ve Xerox'ta Smalltalk'u geliştirenler tarafından yazılan, bir dizi yazı ile, Smalltalk'un kapıları dünyaya tam anlamıyla açıldı.[35] Nesneye yönelimi oluşturan kavramlar, ilk defa, uyumlu ve bir bütün halde büyük bir kitlenin kullanımına sunuldu.

---

[34] Goldberg, A. ve Rabson, D. s. 78

[35] Savic, Düsko. **Object Oriented Programming With SmallTalk / V.** England, Ellis Horwood Limited, 1990, s. 107

1970'lerin başında, Xerox'a bağlı Palo-Alto Araştırma Merkezinde (PARC), Öğrenmeyi Araştırma Grubu (Learning Research Group), insanların bilgisayarları daha istekli kullanabilmelerini amaçlayan bir çalışmaya başladı.

1972'de, öğrenmeyi araştırma grubu'nda görevli Alan Kay, Adele Goldberg ve arkadaşları, tümüyle nesneye yönelik bir dil olan Smalltalk-72 uyarlamasını geliştirdiler. Bu çalışma, yeterli düzeyde olmadığı için dış dünyaya açılamadı.

1976'da Smalltalk-76 versiyonu geliştirildi. Eski uyarlamasında bir ASCII terminalini andıran kullanıcı arabirimi geliştirilerek, nokta-grafik tabanlı ve mause ile yönetilen bir kullanıcı arabirimine dönüştürüldü.[34]

Mouse ile yönetilen nokta-grafik tabanlı kullanıcı arabirimi düşüncesi, ilk olarak Apple tarafından üretilen Lisa adlı bilgisayarda boy gösterdi, daha sonra da neredeyse Macintosh bilgisayarlar ile bütünleşti.

1980 yılında Smalltalk son halini aldı ve Smalltalk-80 versiyonu üretildi. Bu uyarlama daha taşınabilir özelliktedir. Bundan önceki Smalltalk uyarlamaları, sadece Xerox üretimi Dorado, Alto ve Dolphin adlı bilgisayarlarda çalışıyordu. 80 uyarlaması, aynı zamanda ilk Smalltalk standardı olarak da kabul edilebilir.

Ağustos 1980'de piyasada bulunan donanımların, Smalltalk için yeterli olmadığının bilincinde olan Xerox, üretilen yazılıma uygun donanıma sahip bilgisayarların üretilmesi ve yaygınlaşması amacıyla Apple, Digital Equipment Corporation, Hewlett-Packard ve Tektronix'ten gözlemciler çağırarak Smalltalk sistemini tanıttı.

Bir yıl sonra 1981 yılı sonlarına doğru ünlü bilgisayar dergisi Byte'ta yer alan ve Xerox'ta Smalltalk'u geliştirenler tarafından yazılan, bir dizi yazı ile, Smalltalk'un kapıları dünyaya tam anlamıyla açıldı.[35] Nesneye yönelimi oluşturan kavramlar, ilk defa, uyumlu ve bir bütün halde büyük bir kitlenin kullanımına sunuldu.

---

[34] Goldberg, A. ve Rabson, D. s. 78

[35] Savic, Düsko. **Object Oriented Programming With SmallTalk / V.** England, Ellis Horwood Limited, 1990, s. 107

1983 yılında, Smalltalk'ın önde gelen tasarımcılarından Adele Goldberg ve David Robson tarafından yazılan "Smalltalk-80, The Language and it's Implementation" adlı kitap yayınlandı. Bir ders kitabı olabilecek kadar eğitsel, pek çok atıf yapılabilecek kadar bilimsel ve iyi bir Smalltalk programcısının bile bilmediği bilgileri içerecek kadar detaylı olan bu kitap, Smalltalk'ın ve nesneye yönelimin yaygınlaşmasında önemli rol oynadı.[36]

1983 yılında, AT & T'den Bjanre Stroustroup C++'i geliştirdi. Ayrıca daha önce Pascal olarak Nuklaus Wirth tarafından geliştirilen, Borland firması tarafından Borland Turbo Pascal halinde yeniden geliştirilmiştir.

Gerek Borland C++, gerekse Borland Turbo Pascal, Smalltalk programlama dilinden sonra geliştirilen ve nesneye yönelime uygun programlama dilleri olarak, günümüzde en yaygın ve etkin kullanılabilen dillerdendir.[37]

2.2. NESNEYE YÖNELİMİN TEMEL KAVRAMLARI VE ÖZELLİKLERİ

Nesneye yönelim tekniği, kapsamının genişliği ve sağladığı avantajlar nedeniyle bir çok kavrama sahiptir. Nesneye yönelik yaklaşımın üstün olmasının ve tercih edilmesinin en büyük sebebi bu kavramların çokluğu ve özelliklerinin fazlalığıdır.[38] Birbirlerini destekleyen bu kavramların hepsi kendine göre öneme sahiptir.

En temel kavram, yaklaşımın isminde de yer alan nesne kavramıdır. Programlamada oluşturulan mantık da, nesne kavramı dikkate alınarak oluşturulmuştur. Ancak sadece nesne kavramının olması, elbetteki nesneye yönelik yaklaşımın fonksiyonlarının anlatılması için yeterli değildir. İşlemlerin her bir nesne üzerinde teker teker yapılmaya çalışılması, bu yaklaşımın avantajlarını yok eder. Bu nedenle, nesneleri bir araya toplayacak, onlar üzerinde toplu işlemler yapılmasını sağlayacak diğer temel kavramlara da ihtiyaç vardır. İşte bütün bu kavramların tümü, nesneye yönelik yaklaşımın

---

[36] Heintz, Timothy, J. "An Object-Oriented Approach to planning and Managing Software Development Projects". **Information & Management.** New York, McGraw-Hill, 1991, ss. 78-79

[37] Smedema, C. H. **The Programming Languages Pascal,Modula, CHILL, and Ada.** Englewood Cliffs, Prentice-Hall, 1983. s. 55

[38] Rumbaugh, J. **Object-Oriented Modeling and Design.** U.S.A., Prentice-Hall, 1990. s. 32

mantığını oluşturmaktadır. Bu kesimde, bu yaklaşımla ilgili kavramlar, kavramlar arası ilişkiler ve yaklaşım özellikleri ele alınmaktadır.

2.2.1. Nesne (Object)

Gerçek dünyada karşılaştığımız, fonksiyonları olan bütün varlıklar, nesneye yönelik yaklaşım tekniğinde ve buna bağlı olarak geliştirilen nesneye yönelik programlamada nesne olarak kabul edilmektedirler. Nesneler canlı veya cansız durumda, soyut veya somut olabilen, ve en az bir özelliği bulunan varlıklardır. Herhangi bir masa, sandalye, telefon, bilgisayar, elma, nokta veya virgülden her biri, birer nesnedirler. Örneğin bunlardan, elma nesnesi ele alınarak incelenebilir. Bir elmanın yazılım terimleri ile ifade edilmesi mümkündür. Yapılması gereken ilk iş, elmanın özellikleri dikkate alınarak ayrıştırılmasıdır.

K'nın elmanın kabuğunu, S'nin elmanın taşıdığı sıvıyı, M'nin içindeki meyvesini, Ç'nin çekirdeklerini ifade ettiği düşünülebilir ve bu özellikler dikkate alınarak farklı sınıflar oluşturulabilir. Bu sınıfların hiçbirisi artık elma anlamında değildir. Bu sınıflarda, K, S, M ve Ç özelliklerine sahip nesneler vardır.

Bir başka şekilde olay, bir ressam ve boyacı gibi de tasavvur edilebilir. Bir elma resminin yapıldığı ve boyandığı düşünülebilir. Elmanın resmi, elma değildir. O, düzlem üzerinde oluşturulan bir semboldür. Ancak bütün haldedir. Sadece kabuğu, sıvısı, meyvesi veya çekirdeği boyanmış olsa, bütün bir elma resminden bahsedilemez. O zaman K, S, M, ve Ç nesnelerinin resminden bahsedilebilir. Elma nesnesini elde edebilmemiz için, yazılım için kodladığımız K, S, M ve Ç sınıflarını birbirleriyle ilişkilendirmesi gerekmektedir.[39] Sınıflar, ortak özelliğe sahip nesneleri bir arada bulundururlar ve birbirleriyle olan ilişkileri sayesinde yeni nesnelerin elde edilmesine imkan sağlarlar.

Nesnenin tanımı, farklı şekillerde yapılabilmektedir:

Nesne, veri ile o veri üzerinde tanımlı bazı işlemlerden oluşan bir bütündür. Örneğin, 1 sayısı kendi başına bir anlam taşımaz. Sadece +, -, /, *

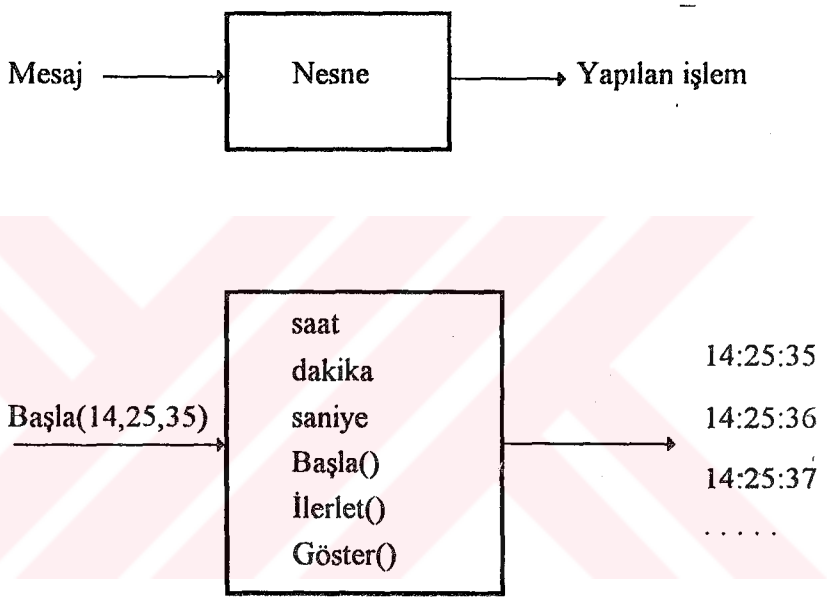
---

[39] Tello, Ernest, R. **Object Oriented Programming for Artificial Intelligence.** Newyork, Wesley Publishing Company, 1989, s. 93

gibi kendi üzerinde tanımlı işlemler ile beraber anıldığı zaman, nesne olma niteliğini kazanır.[40]

En basit anlamıyla bir nesne, mantıksal bir varlıktır. Programlama aracı olan nesne ise, verileri ve bu veriler üzerinde işlem yapmaya yarayan program kodunu içeren bir varlıktır. Programlama açısından nesneler, bir kara kutu olarak değerlendirilir. Nesnelere mesaj göndermek ve bu mesajlarla yapılan işlemlerin sonuçlarını görmek mümkündür.[41]

Nesnelerin nasıl değerlendirildiği ve örnek kullanılışı şekil 2.1'de gösterilmektedir.



Şekil 2.1. Nesne ve Kullanılışı

Şekilde, saat, dakika ve saniye nesneleri bir sınıfa ait olarak tanımlanmaktadır. Bu nesnelere, Başla() fonksiyonu ile mesaj gönderilerek başlangıç değerleri verilmekte, İlerlet() fonksiyonu ile mesaj gönderilerek zaman çalıştırılmakta ve sonuçların gösterilmesi Göster() fonksiyonu tarafından gönderilen mesajlar ile sağlanmaktadır. Nesne içinde nelerin olduğu, dışarıdan görülmemektedir. Nesne içinde bir kısım veri veya program kodu, özel (private)

---

[40] Paepcke, Andreas. **Object Oriented Programming the Clos Perspective.** USA, Teach Books, 1993, s. 45

[41] Hunt,V,Daniel. **Artificial Intelligence & Expert Systems Sourcebook.** Newyork, Chapman & Hall, 1986, s. 183

kısımda yer alır. Özel kısım, herhangi bir sınıfdaki nesne ve fonksiyonlara sadece o sınıfa ait nesne veya fonksiyonlar tarafından erişilebilen ve kullanılabilen kesimdir.[42] Bu kesime diğer sınıflardan erişilemez ve bu kesimdeki nesne veya fonksiyonlar diğer sınıflar tarafından kullanılamaz. Ancak istenirse nesneler veya fonksiyonlar genel (public) olarak da tanımlanabilir. Bu durumda diğer sınıflardan da erişim ve kullanım mümkün olacaktır.

Ayrıca nesnelerin sadece kendisinden türeyen nesnelere, kendilerine erişmeleri ve kullanmaları için müsade etmeleri durumunda korunmuş (protected) kesim oluşur. Yani korunmuş kısımda yer alan elemanlara, herhangi bir nesnenin kalıtım (inheritance) yoluyla türeretilen alt nesneleri tarafından erişilebilir.[43] Program kodu ve verinin, bir nesne içinde bu şekilde birbirine bağlanmasına paketleme (encapsulation) denir. Özel ve korunmuş kısımlarla sağlanan bu korunma, istendiğinde arkadaş (friend) kullanım özelliği ile kaldırılabilir.

Nesneler kullanıldıkları yerlere göre, dört kategori halinde incelenebilir:

1. Denetim : Nesneler bir programın normal akışı sırasında, bilgi işlem denetim elemanı olarak görev yapabilirler. Program içinde çalışacak olan sistem, bir nesne ile gösterilir ve denetim bu nesnenin fonksiyonları ile sağlanır.

2. Uygulama : Bir sistemde belirli amaçların gerçekleştirilmesi için nesneler kullanılabilir. Çeşitli varlıkların bilgisayar ortamında gösterimi nesnelerle yapılabilir. Bu da daha kolay bir soyutlama sağlar.

3. Veri Yönetimi : Üzerinde işlem yapılacak veriler, nesne haline getirilir. Bu nesneler üzerinde yapılan işlemler metod olarak tanımlanır. Nesneye yönelik veri tabanı yönetim sistemi buna bir örnektir.

4. Arabirim : Kullanıcı ile uygulama arasındaki arabirim nesnelerle sağlanabilir. Bu da her iki düzey arasında bağımsızlık sağladığından, herhangi

---

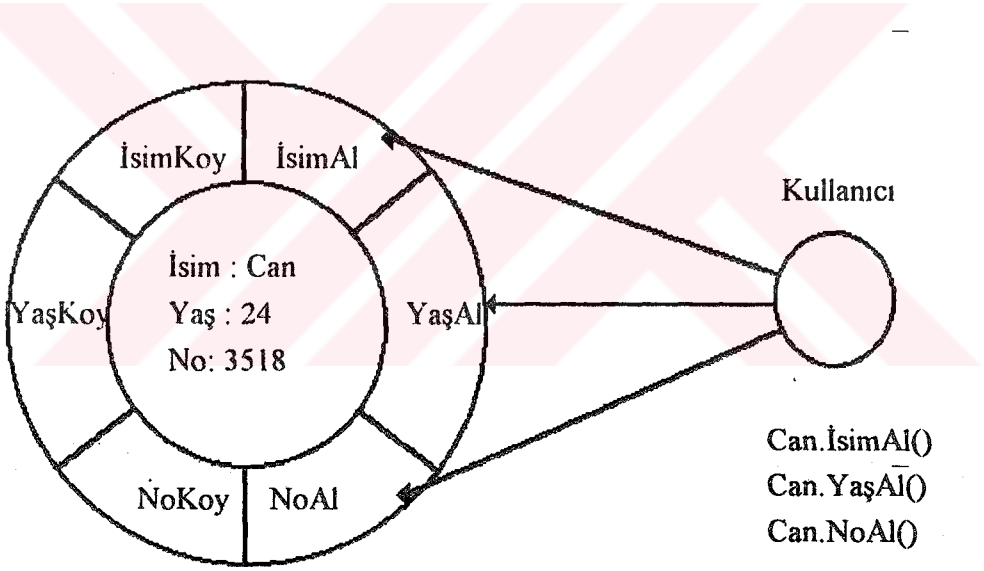
[42] Morris, Stephen. **Object Oriented Programming Under Windows.** British Library, Butterworth - Heinemann Ltd, 1994. s. 21

[43] Atkinson, Colin. **Object Oriented Reuse Concurrency and Distribution.** Newyork, ACM Press, 1991. s. 79

birinde yapılacak değişiklik veya gelişim diğerini de etkileyecek problemler çıkarmaz.[44]

Yazılım dünyasındaki nesneler, gerçek dünyadaki varlıklar gibi davranış gösteren program parçaları ve verilerdir. Bir nesnede onu tanımlayan bilgiler, veriler ve bunlar üzerinde işlem yapmaya yarayan metodlar bulunur.[45] Nesnelerin en büyük avantajı yeniden kullanıma açık olmalarıdır. Böylelikle önceden hazırlanmış, test edilmiş ve çalışır hale gelmiş programlar daha sonra çok kısa sürede yeni bir programa uyarlanabilir.

Nesneye yönelik programlamanın, temel birimi nesnedir. Yapısal programlamada yapılar ve işlevler ayrı ayrı ele alınırken, nesneye yönelik programlamada veriler ve işlevler nesne ile birleştirilir.[46] Bir nesne, verileri gizleyerek programlama dillerinin ve uygulamaların temel prensiplerinden olan veri saklama ilkesini destekler. Bu verilere erişim o nesneye ait üye fonksiyonlarla sağlanır.[47]



Şekil 2.2. Üye Fonksiyonlarla Nesne Kullanımı

---

[44] Sarıdoğan,M,Erhan. **C++ ve Nesneye Yönelik Programlama.** İstanbul, Sistem Yayıncılık, 1994, ss. 87-88

[45] Demurlian, S, A. Beshers, G, M. ve Ting,T,C. "Programming Versus Databases in TheObject-Oriented Paradigm". **Information and Software Technology.** No:2. USA, Butterworth Ltd, 1993, s. 58

[46] Voss, Greg. **Object Oriented Programming: An Introduction.** Tokyo, Osborne McGraw - Hill, 1991, s. 63

[47] Wegner, P. "Clasification in Object-Oriented Systems". **Proceeding of the Object-Oriented Workshop.** New York, Sigplan, 1986, s.124

Şekil 2.2'de görüldüğü gibi üye fonksiyonlar nesnenin iç yapısını gizlemekte ve bir kabuk oluşturmaktadır. Verilere ulaşmak için arabirim kullanılmalıdır. İsim, Yaş ve No nesnelerine İsimKoy, İsimAl, YaşKoy, YaşAl, NoKoy ve NoAl üye fonksiyonları ile mesajlar gönderilmektedir.

Nesne, belleğin değiştirilebilen değerler veya belirlenmiş fonksiyonlarını yerine getiren isimlendirilmiş bir bölgesidir. Bu yaklaşımda tüm değişkenler birer nesnedir. Nesnelerin davranışlarına göre tasnif edilmeleri de, sınıf (class) kavramını ortaya çıkartır.

Nesneler, değişken ve fonksiyon içeren yapısal değişkenlerdir. Nesnelerin içerdikleri bu değişkenlere üye değişken (member variable), fonksiyonlara da üye fonksiyon (member function) adı verilir.[48]

Bir nesnenin üye değişken ve fonksiyonlarının diğer nesneler tarafından doğrudan kullanılmasının istenmediği durumda, bu üyeler diğer nesnelere karşı korunabilirler.

Bir nesnenin, diğer nesneler veya fonksiyonlar tarafından kullanılabilmesi bazı sınırlamalara bağlıdır:

1. Bir nesnenin veya fonksiyonun yalnız üyesi bulunduğu nesne içerisinde kullanılması halinde, özel (private) nesne veya fonksiyon tanımı yapılmış olur. Nesne ve fonksiyonlar özel durumdadır.

2. Bir üyenin içinde bulunduğu nesne haricinde ancak bu nesneden türeyen yeni nesneler tarafından da kullanılabilmesi durumunda korumalı (protected) üye oluşur. Bu durumda, sözkonusu nesneden türemeyen nesnelerin ve fonksiyonların erişimleri mümkün değidir.

3. Bir üyenin, tüm nesneler ve fonksiyonlar tarafından ortaklaşa kullanılabilmesi ise ortak (public) üye kullanımı olarak adlandırılır.

4. Bir nesnenin başka bir nesneyi arkadaş ilan ederek üyelerinin tümünün bu nesne tarafından kullanılmasına izin vermesi halinde arkadaş (friend) nesne özelliği gerçekleşir.[49]

---

[48] Caruso, M. ve Sciore, E. **The Vision Object-Oriented Database Management System.** Newyork, ACM Press, 1987, s. 109

Bir nesne (object), bilginin özelliğini ve davranışını, yani işlemi tanımlar. Tablo 2.1'de verilen örnek ile, nesne yapısı ve kullanımı gösterilmektedir. Bu örnekte, Ucak adında bir obje tanımlanmış ve fonksiyonlar vasıtasıyla bu objeye özellikleri tanıtılmıştır.

```
Type
        Ucak=Record
        Hız:Word;    Yükseklik:Word;    Kanatlar:(Yukarı,Asagı);
End;
{ Uçağın hareketleri şu prosedürler ile tanımlanabilir:}
Procedure Hızlanma
        Begin
        .....
        End;
Procedure Yavaslama
        Begin
        .....
        End;
Procedure KanatKaldır
        Begin
        .....
        End;
Procedure KanatİndirBegin
        .....
        End;
```

Nesneye yönelik programlamada bilgilerin özelliği ve davranışı tek bir yapıda, yani nesne (object) de toplanır.[50]

```
Type
        Ucak=Object
        Hız:Word;    Yükseklik:Word;    Kanatlar:(Yukarı,Asagı);
        Procedure İlkislemler;
        Procedure Hızlanma;
        Procedure Yavaslama;
```

---

[49] Demiralp,Fehmi. **Turbo ve Borland C++ İle Nesneye Dayalı Programlama.** İstanbul, Beta Basım Yayım, 1993, ss. 39-40

[50] Çubukçu,Faruk. **Turbo Pascal Programlama Dili.** İstanbul, Türkmen Kitabevi, 1991, ss. 201-203

```
        Procedure Yükselme;
        Procedure Alcalma;
        Procedure KanatKaldır;
        Procedure KanatİndirI;
End;
```

Bu tanımlamanın ardından, nesnenin kullanımı ise şu şekildedir;

```
Procedure Ucak.İlkislemler;
        Begin
                Kanatlar:=Yukarı;
                Hız:=0; Yüseklik:=0;
        End;
        Var
                N:Ucak;
                With N do
                        Begin
                                İlkislemler;    KanatKaldır;
                                Hızlanma;       Yükselme;
                        End;
```

Tablo 2.1. Nesne Yapısı ve Kullanımı

2.2.2. Sınıf (Class)

Aynı özelliği taşıyan nesnelerin, bir arada bulunarak, ortak hareket etmelerini sağlayan nesneler topluluğu, sınıf olarak tanımlanır. Eğer sınıflandırma özelliği olmasydı nesneye yönelik yaklaşım, içinden çıkılması mümkün olmayan bir hal alırdı. Etrafımızda bulunan herhangi bir nesneden söz ederken ya da onu birisine tanıtırken, o nesnenin özelliklerinden bahsederiz. Nesnenin şeklinden, ağırlığından, canlı veya cansız oluşundan, boyundan, renginden vs. Nesneye yönelimde, nesne, özellikleri ve davranışlarıyla bir sınıf yapısının içinde yer alır. Her nesnenin mutlaka üye olduğu bir sınıf söz konusudur.

Bir sınıf, esas itibariyle bir veri yapısına (structure) benzer. Veriler üzerinde yöntem (method) tanımlama işlemi tek tek her veri üzerinde değil, benzer yapısal özellikler gösteren nesnelerin oluşturduğu, sınıflar üzerinde tanımlanır. Her nesne, bir sınıfın bireyi olmak zorundadır. Sınıf, aynı yapısal ve davranışsal özellikleri taşıyan bireylerden oluşan bir yapıdır. Sınıflar da aynı

zamanda birer nesnedirler. O halde birer nesne olan sınıfların da bir sınıfı vardır. Sınıflar, bir hiyerarşi içinde yapılanmışlardır. En tepede en genel özelliklere sahip sınıf olan Object yer alır. Sınıf hiyerarşisi adı verilen bu hiyerarşi, bir ağaç yapısına sahiptir. Her sınıf sadece tek bir sınıfın alt sınıfıdır.[51]

Nesneye yönelimin tercih edilmesinin en önemli özelliği, sınıflama sistemidir. Bir programı birimler halinde (module) yazmak, belirli bir veri tipine ait tüm bilgilerin, merkezi bir birimin kontrolünde olmasını sağlar.

Bir grafik sisteminde kullanılmak amacıyla resim sınıfını tanımladığımızı ve sistemin sadece daire, üçgen ve kare şekillerini desteklemesi gerektiğini varsayalım. Ayrıca nokta ve renk sınıflarının da daha önce tanımlandığını kabul edelim. Bir resim sınıfı, sınıflama özelliği kullanılarak Tablo 2.2'de gösteridiği gibi tanımlanabilir ve kullanılabilir:

```
enum sec {daire, ucgen, kare }
class resim {
        point nokta;
        color renk;
        sec k;
    public:
        point where()    {dönüş merkezi}
        void move (point to)
        void rotate (int);
        // diğer işlemler
        };
```

Tablo 2.2. Sınıf Tanımı ve Kullanımı

Bu tanımlamada, resim sınıfına ait where(), move() ve rotate() adlı fonksiyonlar yardımıyla ve k adlı alanın değerine göre hangi şeklin çizileceği tesbit edilir.

---

51 Tom, Gilb. **Principles of Software Engineering Management.** New York, Addison-Wesley, 1988, s.106

Bir sınıf private (özel) ve public (genel) kısımlara sahip olabilir. Sınıf içinde tanımlanan her eleman otomatik olarak (default), özel durumdadır. Yani özel bir alan tanımı için, private ifadesinin kullanılması mutlaka gerekli değildir. Özel ve genel alan tamımlamaları Tablo 2.3'de gösterilmektedir.

```
Class Kuyruk {      int i[100];
                    int eleman;
              public :
                    void bos(void);
                    void doldur(int i);
                    int cıkar(void);
                    int eleman_sayısı();
              };
```

Tablo 2.3. Sınıflamada Kullanılan Alanların Tanımlanması

Kuyruk adlı sınıf içerisinde tanımlanan i ve eleman isimli değişkenler, private sözcüğü kullanılmadığı halde özel durumdadır. Bu kısımlara, o sınıfa ait olmayan hiçbir fonksiyon tarafından erişilemez. Bu durum, nesneye yönelimin diğer temel bir özelliği olan paketleme (encapsulation) kuralının gereğidir. Bazı veri yapıları özel tanımlanarak, başkaları tarafından erişilmeleri önlenir. Aynı şekilde fonksiyonlar da özel olarak tanımlanabilir. Bu tip fonksiyonlar, sadece o sınıfa ait diğer üye fonksiyonlar tarafından kullanılabilir.

Programın, diğer kısımlar tarafından da erişilebilmelerine müsade edilen bölümleri, genel (public) olarak tanımlanmalıdır. Sınıf içinde public sözcüğünden sonra tanımlanan her şeye, sınıf dışından erişilebilir.[52] Genel olarak, sınıf üyelerinin public olarak tanımlanması sınırlı tutulmalıdır. Hatta gerekmedikçe kullanılmamalıdır. Nesneye yönelimin ilkeleri gereği bütün veriler özel (private) yapılmalı, bu verilere ulaşımlar ise genel (public) fonksiyonlarla gerçekleştirilmelidir.

Bundan başka, sınıflarda bir de korunmuş (protected) kısımlar vardır. Bu kısım protected sözcüğü ile belirlenir. Korunmuş veri ve fonksiyonlara yalnızca o sınıftan türetilen alt sınıflar doğrudan erişebilir. Diğer sınıflar erişemez.

---

[52] Yoğurtçu, Tendü. "Nesneye Dayalı Çözümleme İçin Yeni Bir Gösterim Biçimi". **Bildiriler.** İstanbul, Tayf Matbaacılık, 1992, s. 274

Tablo 2.3'de yer alan, bos(), doldur(), cıkar() ve eleman_sayısı() fonksiyonlarına üye fonksiyonlar (member functions) denir. Sınıf içinde tanımlanan fonksiyonlar, o sınıfın özel (private) kısımlarına erişebilirler

Sınıf tanımlanmasının ve kullanılmasının genel şekli, nesneye yönelik programlama dillerinden C++ programlama dilinde, Tablo 2.4'de gösterildiği gibidir.[53]

```
Class SınıfAdı    {
                        //Özel veriler ve fonksiyonlar
                        public:
                        //Genel veriler ve fonksiyonlar
                        protected:
                        //Korunmuş veriler ve fonksiyonlar
                } nesneListesi;
```

Tablo 2.4. Sınıf Tanımlamaları

Hiçbir tanımlama yapılmadan kullanılan nesneler ve fonksiyonlar özel (private) tanımlanmış kabul edilirler. Public ifadesi ile kullanılan nesne ve fonksiyonlar ise genel (public) tanımlıdır ve bütün nesne ve fonksiyonlar tarafından erişilebilirler. Protected ifadesi ile kullanılan nesne ve fonksiyonlar ise, sadece o sınıf tarafından türetilen nesne ve fonksiyonlar tarafından erişilebilinen korunmuş (protected) kısımda bulunan nesne ve fonksiyonlardır.

2.2.3. Kalıtım (Inheritance)

Sınıflama, nesneye yönelimde önemli bir özellik olmasına rağmen, nesneye yönelik yaklaşımın fonksiyonlarının gerçekleştirilmesi için yeterli değildir. Sadece sınıflama özelliği ile, nesneye yönelimde istenilen esneklik elde edilemez.

Nesneye yönelimde bir sınıfı, diğer bır sınıfa altsınıf (subclass) olarak tanımlamak mümkündür. Zaten yeni bir sınıf tanımlanırken, daha önce

---

[53] Sarıdoğan,M,Erhan. ss. 125-127

tanımlanmış sınıflardan hangisinin ya da hangilerinin alt sınıfı olacağı da belirtilir. Her sınıf, altsınıfı olduğu sınıflara ait özellikleri kazanır.[54]

Kalıtım (inheritance), sınıflar arasında, birbirinin özelliklerini kazanma ilişkisine verilen addır.[55]

Genellikle iki şekilde kullanılır. Birincisi, verilerin kapsanması maksadıyla sınıfların özelliklerine göre hiyerarşik bir yapı oluşturulması, ikincisi de, tekrar kullanma imkanını sağlamak amacıyla bir öncekine aynen benzeyen bir sınıf oluşturulmasıdır. Her iki şekilde de bir temel sınıf, özelliklerının tümünü veya bir kısmını bir başka sınıfa aktarır.

Kalıtım hiyerarsisinin başında bulunan sınıfa temel sınıf, bazen de üst sınıf (baseclass) denir. Bu sınıfın özelliklerini alarak gelıştirilen yeni sınıfa da türetilmiş sınıf veya alt sınıf adı verilir. Bu türetme işlemi, istenildiği kadar yapılabilir. Fakat pratikte üç veya dördü geçmez.

Kalıtım, mevcut bir sınıfın biraz daha geliştirilerek yeni bir sınıf oluşturulması olarak da açıklanabilir. Temel sınıfın, tam veya bir kısım özelliklerinin kullanılacağı, ancak yeni bir sınıfın da gerekli olduğu durumlarda, kalıtım kuralının kullanılması, programcılıkta çok etkili bir yöntem olarak kabul edilmektedir.

Daha önce de belirtildiği gibi, kalıtım iki şekilde ve iki farklı amaçta kullanılır. Bunlardan hiyerarşik yapı oluşturarak gerçekleştirilen kalıtıma, genel kalıtım (public inheritance), tekrar kullanıma imkan sağlamak amacıyla var olan kodun tekrar edilmesiyle oluşturulan kalıtıma ise özel kalıtım (private inheritance) denir.

Kalıtım bir başka tasnif şekline göre de, basit kalıtım ve çoklu kalıtım şeklinde, iki kısımda incelenebilir:

---

54 Halbert, D, C. ve O'Brien, P, D. "Using Types and Inheritance in Object-Oriented Languages". **European Conference on Object-Oriented Programming.** Paris, 1987, ss. 89-91

55 Morris,Stephen. **Object Oriented Programming Under Windows.** British Library, Butterworth - Heinemann Ltd, 1994, s.23

1. Basit Kalıtım: Basit kalıtımda yalnızca bir tek temel sınıf bulunur. Türetilmiş sınıf, temel sınıfın tüm özelliklerini taşır. Basit kalıtım sırasında veri ve üye elemanlara erişim bazı şartlara bağlıdır.

Türetilen sınıf, temel sınıfın özel kısımlarına erişemez. Bu sayede türetilmiş sınfın, temel sınıfın veri gizleme ve kapsama özelliğine zarar vermesi önlenir.

Temel sınıfın elemanlarına hem türetilmiş sınıf, hem de diğer sınıflar tarafından kolaylıkla erişilebilmesi için, bu elemanların genel (public) kısımlara koyulması gerekir.[56] Elemanlar eğer korunmuş (protected) kısma koyulursa, bu sefer sadece türetilmiş sınıflarca erişilebilir. Yani basit kalıtımda, türetilmiş sınıflar, temel sınıfların hem genel, hem de korunmuş kısımlarına erişebilirler. Yanlızca özel kısımlarına erişemezler.

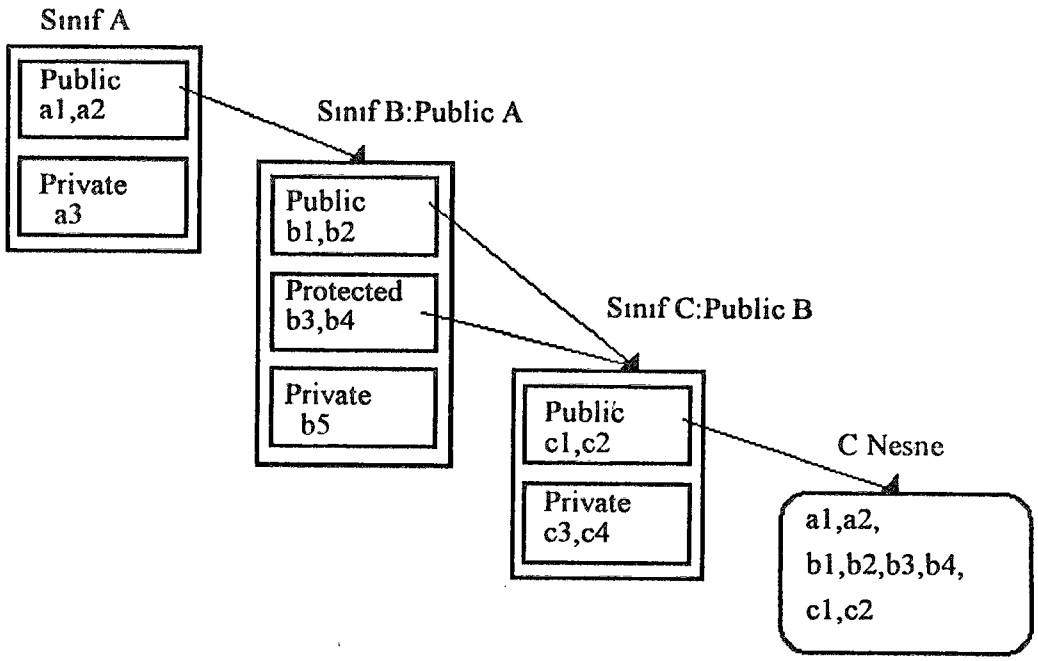
Eğer temel sınıfın özel tanımlı elemanlarına, alt sınıflardan bir kısmının erişmesi isteniyorsa, bu türetilmiş sınıfın, temel sınıfa erişiminde sınıflar arkadaş (friend) olarak tanımlanmalıdır. Friend tanım ile, temel sınıfın özel elemanını korunmuş yapmak aynı manadadır. Tüm alt sınıflar, temel sınıfın korunmuş (protected) kısımlarına erişebilirler.

Basit kalıtımın şeklinde, temel sınıfın genel kısmında bulunan bütün elemanlarına türetilmiş sınıf tarafından erişilebilmektedır. Basit kalıtımın erişim yöntemi şekil 2.3'de gösterilmektedir. Burada, a1, a2, b1, b2, b3, b4, c1 ve c2 elemanlarına, nesne tarafından erişilebilmekte, a3, b5, c3 ve c4 elemanlarına ise erişilememektedir.[57]

---

[56] Voss,Greg. **Object Oriented Programming: An Introduction.** Tokyo, Osborne McGraw - Hill, 1991, s. 33

[57] Sarıdoğan,M,Erhan. ss. 149-153



Şekil 2.3. Basit Kalıtım Erişim Yöntemi

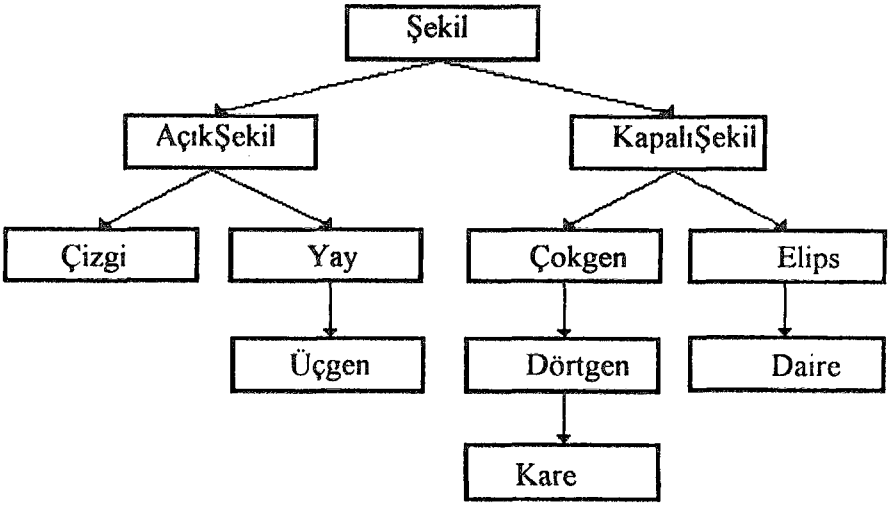
Basit kalıtımda türetilmiş sınıfın elemanları, temel sınıfının genel (public) kısmında bulunan elemanlarına erişebilmekte, özel kısmında bulunan elemanlarına ise, özel kısmın tanımlı olduğu sınıf adı belirtilerek erişilebilmektedir.

2. Çoklu Kalıtım: Tek bir sınıfta yapılan türetim, her zaman yeterli olmayabilir. Özelliklerin, birden fazla temel sınıftan alınmaları gerekebilir. Çoklu kalıtım (multiple inheritance) dediğimiz bu yöntemle çeşitli sınıfların özelliklerini, tek bir türetilmiş sınıfta birleştirerek, karmaşık yapıların bir arada oluşturulmaları sağlanır.[58]

Şekil 2.4'deki şemada, örnek olarak iki boyutun alındığı, grafiksel şekiller üzerinde uygulanan, çoklu kalıtım erişim yöntemi gösterilmektedir. Burada AçıkŞekil ve KapalıŞekil sınıfları, Şekil sınıfına erişebilmekte, bu sınıfın eleman ve fonksiyonlarını kullanabilmektedirler.

---

[58] Stein, L, A. Lieberman,H. ve Ungar,D. "A Shared View of Sharing: The Treaty of Orlando". **In:Object-Oriented Concepts, Databases and Applications.** New York, ACM Press, 1989, s.143



Şekil 2.4.Çoklu Kalıtım Erişim Yöntemi

2.2.4. Paketleme (Encapsulation)

Nesneleri oluşturan veri, değişken ve yordamların, aynı sınıfta toplanmasına paketleme veya kılıflama (encapsulation) denir.[59]

Çevremizdeki nesneleri onların niteliklerini kullanarak tanımlarız. Daha önce de bahsedildiği gibi, nesneye yönelik programlamada nesneler, nitelikleri ve davranışlarıyla bir sınıf yapısını oluştururlar.

Örneğin, bir nesne olarak düzlemde yer alan bir noktayı düşünelim. Bu nokta, Turbo Pascal programlama dilindeTablo 2.5'deki gibi tanımlanabilir:

```
Type
        nokta=object
        x,y=integer;
        renk=char;
        procedure hareket(x,y,renk);
End;
```

Tablo 2.5. Turbo Pascal'da Nesne Tanımlanması

---

59 Bulucu,Feza. **Etkin Modem Haberleşmesi Sağlayan Nesneye Dayalı Bir Yazılım Tasarımı.** İstanbul, Basılmamış Y.Lisans Tezi, 1991, s. 26

Burada, noktanın düzlemdeki yerini ve rengini belirleyen x,y ve renk, nokta nesnesinin nitelikleridir. Hareket adlı prosedür ise, noktanın davranışlarını belirlemektedir. Bu durum, nesnelerin, değişkenlerin ve yordamların aynı sınıf içerisinde bulunmalarını sağlamakta ve nesneye yönelimin paketleme (encapsulation) özelliğini gerçekleştirmektedir.[60] Bu şekilde daha modüler bir yapı oluşturulmakta ve problemlerin anlaşılması ve çözümlenmesi kolaylaşmaktadır.

Bir başka yaklaşıma göre paketleme (encapsulation), kod ve verilerin birlikte objeler içerisinde bir araya getirilmesidir.[61] Turbo Pascal nesneye yönelik programlama dili, özel bir yönlendirici kullanımıyla encapsulation özelliğine imkan sağlar. Burada, nesne sahalarına sadece ilgili nesneye ait prosedürler tarafından erişilebilir. Tablo 2.6'da, sahalar ve metodlar için özel bölümler oluşturulmadan Location ve Point adlı nesnelerin kullanımı gösterilmektedir.

```
type
        Location = object
        X,Y:Integer;
        procedure Init(InitX, InitY:Integer);
        function getX:Integer;
        function getY:Integer;
        end;
        Point = object(Location)
        Visible:Boolean;
        procedure Init(InitX,InitY:Integer);
        procedure Show;
        procedure Hide;
        function IsVisible:Boolean;
        procedure MoveTo(NewX,NewY:Integer);
        end;
```

Tablo 2.6. Turbo Pascal'da Nesnelerin Kullanılması

---

[60] Bancilhon,F. Delobel,C. ve Kanellakis,P. **Building an Object Oriented Database System The Story Of O2.** USA, Morgon Kaufmann Publishers, 1992, s. 7

[61] Morris,Stephen. s. 18

Burada yalnızca üç data sahası vardır; X, Y ve Visible. MoveTo prosedürü X ve Y için yeni değerleri yükler ve getX ve getY fonksiyonları X ve Y nin değerlerini geri getirir. Bu durum, X ve Y'ye direk olarak daha fazla erişimi gerektirmez. Show ve Hide prosedürleri, Visible değişkeninin true veya false almasını sağlar. IsVisible fonksiyonu da Visible değişkeninin o anki değerini döndürür.[62]

2.2.5. Çokşekillilik (Polymorphism)

Çokşekillilik (polymorphism) değişik yapılara sahip fonksiyonların, aynı isimle kullanılmaları işlemine verilen addır.

Bir başka ifade ile, birbirinden türeyen sınıflarda benzer davranışları belirlemek için yazılan yordamlar aynı isimde olabilirler. Bu yordamlar programın çalışması sırasında hangi sınıfla ilgili olarak çalışmışsa, onunla ilgili bir işlevi yerine getirir.

Nesneye dayalı programların en önemli özelliklerinden biri olan çokşekillilik (polymorphism) sayesinde, aynı isimdeki bir yordam ile, değişik varlıkların davranışlarını belirtmek mümkün olmaktadır.[63]

Birbirlerinden türeyen sınıflarda yer alan varlıkların davranışları da benzer olamaktadır. Ancak programın çalışması sırasında, herhangi bir anda bu varlıklardan hangisinin kullanılacağı belirsiz olabilir. Bu durumda, sınıfların davranışlarını belirleyen yordamlara aynı isim verilir ve belli bir anda hangi yordamın çağrılacağına program çalıştırılırken (run time) karar verilir. Çokşekillilik kavramı bir örnek üzerinde Tablo 2.7'de gösterilmektedir.

```
Class daire
        { protected:
              int merkez_x,merkez_y,yarıçap;
          public:
              virtual void ciz();
         };
Class silindir:public daire
        {     int yükseklik;
```

[62] Borland. **Turbo Pascal User's Guide.** USA, Borland International, 1990, ss. 88-89
[63] Florentin,J,J. s.29

```
public:
    void ciz();
};
```

Tablo 2.7. Çokşekilliliğin Uygulanışı

Görüldüğü gibi, her iki sınıfta da ciz isimli bir yordam bulunmaktadır. İsimleri aynı olmakla birlikte, bu yordamların iç yapıları ve yaptıkları işler farklıdır. Bu yordamların çağırılma şekillerini izah edebilmek için, işaretçilerden (pointerlerden) bahsetmemiz gerekmektedir. Pointer kullanımı, nesneye yönelik programlamanın vazgeçilmez özelliklerindendir. Programcıya esnek programlar yazmada kolaylıklar sağlar. Her değişkenin bellekte bir adresi vardır. Değişkenlerin isimleri verilmeden, doğrudan adresleri verilerek bu değişkenlere erişebilinir. Yani pointerler, herhangi bir değişkenin adresini gösterirler. Pointer kullanılmadan değişkenlerin, hafızada (memory) nerede saklandığı bilinemez. Normal değişkenlerde, o değişkenin değeri saklanır. Pointerlerde ise, yalnızca verinin saklandığı adres saklanır. Pointerler kullanılmadan önce tanımlanmalıdırlar ve değerleri atanmış olmalıdır. İki çeşit pointer işlemcisi vardır:

1. &, bellek adresini gösterir.
2. *, adres değerini gösterir.

&, işareti adres işlemcisidir. örneğin X, tanımlanmış bir değişken ise, &X onun değerinin bulunduğu adresi saklar. * işlemcisi ise bu adresteki değeri göstermek için kullanılır.[64]

Pointerler hakkında verilen bu genel bilgilerden sonra, çokşekillilik kavramı için Tablo 2.7'de verilen örnek üzerinde, önceden tanımlanmış bulunan daire ve silindir sınıflarına işaret eden pointerler tanımlanabilir.

```
Daire *işaret_daire;
Silindir *işaret_silindir;
Bu iki tanımlamadan sonra atamalar yapılabilir.
işaret_daire=işaret_silindir;
```

---

[64] Li, S. ve Shahidehpour, S, M. "An Object Orienfed Power System Graphics Package for Personal Computer Environment". **IEEE Transactions on Power Systems.** No:3. USA, IEEE, 1992, ss. 81-83

Kural olarak alt sınıf pointer'i, ana sınıf pointer'ine atanabilir. Çünkü alt sınıf daha çok veri içermektedir. Bu sayede, ana sınıfa ait veri alanları doldurulabilir. Ancak ana sınıf pointer'i alt sınıf pointer'ine atanamaz. Bu atama şekli ile yordamların çok şekilli olması sağlanabilir. Bu durum Tablo 2.8'de verilmektedir.

```
Daire *işaret_daire, nesne_daire;
Silindir nesne_silindir
İf (getch()= = 'd') işaret_daire=& nesne_daire;
        Else işaret_daire=&nesne_silindir
işaret_daire Çiz();
```

Tablo 2.8. Pointer Kullanımı İle Çokşekilliliğin Sağlanması

Programın başında daire ve silinidir sınıflarına ait iki nesne tanımlanıyor. Bunlar nesne_daire ve nesne_silinidir. İşaret_daire adlı pointer ise her iki sınıf için de kullanılabilecek durumdadır. Programda kullanıcının girdiği karakter 'd' ise bu pointere nesne_daire'nin adresi, aksi halde nesne_silindir'in adresi aktarılmaktadır. Böylece işaret_daire ciz() satırında gerçekte hangi ciz() yordamın çağrıldığı bilinmektedir.

Görüldüğü gibi, programın derlenmesi esnasında ilgili satırda hangi yordamın çalışacağı belli olmamasına rağmen, programın icrası sırasında hangi yordamın çalışacağı belirlenebilir.[65]

Bu yapı sayesinde oldukça esnek ve verimli programların yazılması mümkün olmaktadır.

2.2.6. Erken ve Geç Bağdaştırma (Dynamic and Late Binding)

Nesneye yönelimin temel özelliklerinden olan, erken bağdaştırma ve geç bağdaştırma, programlama mantığında derleme zamanı ve çalıştırma zamanı olaylarına karşılık gelir.

Nesneye yönelim terimi olarak, erken bağdaştırma, bir nesnenin kendisini çağıracağı fonksiyonla bağlantısının, derleme anında yapılması

---

65 Blair,G. Gallagher,J. Hutchisaon D. ve Shepherd,D. **Object Oriented Languages Systems and Applications.** Newyork, Halsted Press, 1988, s.80

anlamındadır. Bu da, hangi fonksiyonun çağrılacağının, programın derlenmesi sırasında belirlenmesi demektir. Buna örnek olarak standart fonksiyonları ve operatör fonksiyonlarının çağrılmasını verebiliriz. Erken bağlanmanın asıl faydası etkin bellek kullanımıdır.[66]

Nesneye yönelik diller, güç ve esnekliklerinin önemli bölümünü geç bağdaştırmaya (late binding) borçludurlar. Geç bağdaştırma kabaca, bir programın hangi nesne üzerinde hangi yöntemi kullanılacağının çalışma anında belirlenmesidir. Nesneye yönelik olmayan dillerden birinden bir fonksiyon veya alt programa çağrı yapıldığında, derleyici derleme anında hangi rutini çağıracağını kesinlikle bilir. C++ veya Turbo Pascal gibi nesneye yönelik bir dilde ise çağrılacak olan kod nesnenin sınıfına bağlıdır. Bu durum, genellikle hangi rutinin çağrılacağını ancak çalışma anında belirleyebilmemize izin verir.[67]

Bağdaştırma, bir ad ile bir nesne arasında ilişkinin kurulmasıdır. Yani programlama dilinde mevcut bir tanıtıcı ile, değişkenler ya da üniteler arasında kurulan ilişkidir. Derleyiciler sayesinde programcılar, bu ilişkinin derleme ve link etme sırasında yapılıp bittiğini düşünürler. Bir değişken veya fonksiyona verdiğimiz adın, tanımlandığı blok içerisinde hep ona ait olduğunu düşünürüz. Hemen hemen, nesneye yönelik olmayan bütün diller böyle çalışırlar. Bu diller, aslında erken bağdaştırma mekanizmasına sahiptirler. Çünkü adlandırma, program çalışmadan önce yapılır. Buna karşılık, yorumlama esasına göre çalışan pek çok dil, bir programın çalışma anında tanımlanabilmesine izin verir. Hatta nesneye yönelik programlama dillerinden Smalltalk, bu işi çok daha ileri götürerek, programcının çalışma sırasında işletim sistemini değiştirmesine bile izin vermektedir. Bu dilde var olan bir sınıf değiştirildiğinde, yeni sınıf, değişiklikler dışında tüm özelliklerini yukarıdan miras alabilir. Üzerinde çalışılan nesneler, işletim sistemine ait olsa bile sistem çalışmaya devam eder. Eski sınıfa ait tüm nesneler, yenileri ile değiştirildikten sonra, eski sınıf tümü ile devre dışı bırakılabilir. Bu durumlar, bir fonksiyon her çağrıldığında bir tablodan adına karşılık gelen kodun bulunup çalıştırılmasını sağlayan, geç bağdaştırma sayesinde gerçekleşir.

---

66 Bancilhon,F. Delobel,C. ve Kanellakis,P. s. 11
67 Sarıdoğan,M,Erhan. s. 216

Geç bağdaştırmanın yararları, nesneye yönelimin diğer temel özelliği olan çokşekillilik (polimorphism) kavramı incelendiğinde de iyice anlaşılmaktadır.

Farz edelimki nokta, daire, dikdörtken gibi çeşitli nesneleri ekrana çizmeyi sağlayacak bir grafik paketi hazırlıyoruz. Bu iş için, bu çizimlerin ortak özelliklerinin tanımlandığı bir taban sınıf tanımlanmalıdır. C++ veya Turbo Pascal'da gerekli tanımlamalar Tablo 2.9'da gösterildiği gibi yapılabilir:

```
C++
Class Graphobj{
        int x,y; /* Nesne yeri */
      public:
      int getx {return x;};
            int gety {return y;};
      Void moveTo (int newx,int newy)
            {x=newx;y=newy;};
};
Turbo Pascal'da
      Type Graphobj=object;
          x,y:integer;  {Nesnenin yeri}
      Function getx:integer;
      Function gety:integer;
      Procedure moveTo(newx,newy:integer);
        End;
```

Tablo 2.9. Nesnelerin Ortak Tanımlanması

Erken bağdaştırmada program daha hızlı çalışır, buna karşılık programcıya sağlanan esneklik azalır. Geç bağdaştırmada ise nesnenin hangi fonksiyonu çağırdığı, çalıştırma zamanında belirlenir. Bu özellik programcıya geniş esneklik getirir. Ayrıca tekrar kullanılabilen ve sonradan geliştirilme imkanı olan kütüphanelerin oluşturulmasına yardımcı olur.

Programlamanın, bu iki bağdaştırma türünden hangisini kullanacağı, tasarım amacına bağlıdır. Küçük hacimli programlar genellikle erken bağdaştırmayı kullanırken, daha geniş programlar her iki metodu birden kullanırlar. Geç bağdaştırma nesneye yönelik programlamanın üstün

yönlerinden biri olamasına rağmen, bu özellik programların biraz yavaş çalışmalarına neden olur. Fakat bu göz ardı edilebilir bir yavaşlamadır. Bu nedenle de geç bağdaştırma, nesneye yönelik programlamada genellikle en çok tercih edilen bağdaştırma yöntemidir.

Derleyicinin, bir sınıf ile o sınıftan türetilen değerler arasındaki farkı anlayabilmesi ve farklı sınıflarda tanımlanan kodları, ayrı ad kullanarak çağırabilmesi çokşekilliliğin, geç bağdaştırma kullanılarak elde edildiği güzel bir örneğidir.[68]

Geç bağdaştırma programın bellek ihtiyacını artıracaktır. Bunlara rağmen geç bağdaştırma, esnekliği nedeniyle programcının işini oldukça kolaylaştırır. Nesnelerin kendilerini düşünmeleri, birbirlerini takip etmeleri, programcının onları düşünüp takip etmesinden daha iyidir. Ayrıca, diğer programcıların kolayca kullanabileceği yazılımlar hazırlamak için de uygundur. Zaten piyasalarda geliştirilen paket uygulamalar, geniş ölçüde bu tür araçları kullanmaktadırlar.

Son olarak da, geç bağdaştırma yöntemi ile işin çoğu derleyiciye yüklenir. Çünkü geç bağdaştırmada, hangi nesne üzerinde hangi yöntemin kullanılacağı ve hangi rutin'in çağrılacağı çalışma anında tesbit edilir. Bu da derleyicinin bu bilgiler üzerindeki kontrolünün sürekli olmasını gerektirir. Bu sayede hem programın karmaşıklığı, hem de hata oranı azalmaktadır. Donanımın nisbeten ucuzlayıp, yazılım hatalarının önemli masraf çıkarması nedeniyle ve bu durumun devam etmesi halinde, nesneye yönelim ve onun bir özelliği olan geç bağdaştırma, hiçbir programcının göz ardı edemeyeceği kavramlar olacaktır.

2.2.7. Soyutlama (Abstraction)

Kompleks problemlerin çözümlenmesi esnasında, problemin temel bir grup sahalara bölünmesi ve bölümler üzerinde bağımsız olarak çalışılması genel bir yaklaşımdır. Bu yaklaşım, soyutlama (abstraction) kavramına da uygun düşmektedir. Problemin anlamlı ve orantılı bir şekilde safhalara ayrılması soyutlama kavramının özüdür.[69] Problemlerin çözümünde bilgisayar

---

[68] Bancilhon, F. Delobel, C. ve Kanellakis, P. **Building an Object Oriented Database System The Story Of O2.** USA, Morgon Kaufmann Publishers, 1992, s. 14

[69] Hakman, Sina. ve Üncü, A. Nilgün. "Nesneye Dayalı Sistemler ve Kullanıcı Programları". **Bildiriler.** İstanbul, Tayf Matbaacılık, 1992, s. 213

bilimcilerinin alakadar oldukları en önemli husus, problemin uygun safhalara ayrılması ve gereken cihaz ve tekniklerin hazırlanmasıdır. Bu aşamada, en uygun cihaz ve tekniklerin belirlenebilmesi önemlidir ve bunun için şu problemler çözümlenmelidir:

1. Cihaz ve teknikler, problemi çözebilmek için spesifik problemlere uygun olmalıdır.
2. Cihaz ve teknikler, donanıma göre etkin bir biçimde gerçekleştirilmiş olmalıdır.

Problem çözmeyi kolaylaştırmak amacıyla, uygun soyutlamanın belirlenmesi için birkaç yol düşünülmüştür. Soyutlamalar, bazen özel problem safhaları için geliştirilmişken, bazen de genel problem çözmeyi desteklemek için geliştirilmişlerdir. Yine de bilgisayar bilimcilerinin karmaşıklığı gidermek amacıyla, soyutlamayı cihaz ve tekniklerin geliştirilmesi ve gruplandırılması şeklinde, sınırlandırdıklarını söylemek mümkündür.

Nesneye yönelik soyutlama, özel problem safhalarının oluşturulması olarak kabul edilmektedir. Bu amaca ulaşmak için, muhtemel uygulamalar skalasında özel problem çözümünün prensiplerinin iyi algılanması gerekir. Bu safhaların oluşturulmasında, nesneye yönelik yaklaşımda uygulanabilecek genel prensipler aşağıdaki kesimlerde ele alınmaktadır.[70]

2.2.7.1. Veri Soyutlama (Data Abstraction)

Nesneye yönelik yaklaşımın temel prensiplerinden, soyutlama kavramının ilk prensibi veri soyutlamadır (data abstraction). Veri soyutlama, hem veri, hem de veriyi kontrol etmek için gerekli olan algoritmalardan programcının ayrı ayrı haberdar olmasıdır. Soyutlamanın yapılmadığı durumlarda kullanıcılar, faaliyetlerin detaylarından tam olarak haberdar değildirler. Veri soyutlama, ayrı fakat yakınen alakalı iki yapıyı kapsar:

1. Modullere ayırma (Modularisation): Modularisation, kompleks sistemleri bir grup kendine has (self-contained) modüllere bölme ile alakalıdır. Özel bir kısma ait tüm bilgi, sistemde bu model ile tutulur. Bu durum, bu sistemin bir parçasını icra ettirecek algoritmaların ve veri yapılarının model

---

[70] Blair,G. Gallagher,J. Hutchisaon D. ve Shepherd,D. **Object Oriented Languages Systems and Applications**. Newyork, Halsted Press, 1988, s. 80

tarafından kapsanmasını sağlar. Bu, eğer değişiklik yapılmak zorunluluğu varsa ve problemler çözülecekse en uygun işlemin yapıldığı anlamındadır. Daha temelde, programcıların, problem safhalarını bir grup tanımlanabilir özel içerikli parçalara böldüğü özel tasarım, modularisation kavramı ile açıklanabilir. Bu tasarım yaklaşımı, nesneye yönelik soyutlamanın temelini belirler. Bu, aynı zamanda soyutlamanın da ilk seviyesini oluşturur.

2. Bilgi Saklama (Information Hiding): Modüllere ayırma işleminde detayların kullanıcıdan saklanması bir sonraki soyutlama seviyesini oluşturur. Bilgi saklama ile, korunmuş arayüze (interface) kullanıcı, bir nesne ile ulaşmalıdır. Kullanıcının lokal prosedürler veya veri yapıları gibi iç detayları görmesine izin verilmez.[71] Bu, karmaşıklığın giderilmesinde önemli bir kuraldır.

2.2.7.2. Davranış Paylaşımı (Behaviour Sharing)

Nesneye yönelik soyutlamanın ikinci prensibi, sistemlerin davranış paylaşımını desteklemektir. Davranış paylaşımı, aynı arayüz (interface) setinin birçok kısımda ortak kullanımına müsade etmek anlamındadır. Bu, sistemin esnekliğini arttıran bir durumdur. Davranış paylaşımının en yaygın kullanım şekli, sınıflandırma özelliği kullanılarak gerçekleştirilir. Sınıflama, kısımların bir grubunu genel davranış temeline göre yeniden şekillendirir. Örnek olarak, queue; insert, delete ve print'i destekleyen bir kısım olarak tanımlandığında bütün queue'ler insert, delete ve print davranışlarını paylaşırlar.

2.2.7.3. Gelişme (Evolution)

Programcılıkta ihtiyaçların çok sık değiştiği, bir gerçektir. Belirli bir dönem içerisinde sistemler değişir ve ek fonksiyonlara ihtiyaç duyulabilir. Sürekli değişim gösteren ve gelişen ihtiyaçlar nedeniyle, nesneye yönelik soyutlamada gelişme (evolution) prensibi oluşmuştur.[72] Bu değişim ihtiyaçlarının karşılanabilmesi için, geliştirilecek sistemin hedefleri iyi belirlenmelidir. Gelişme prensibi iki şekilde gerçekleştirilebilir:

1. İhtiyaçların Gelişimi (Requirements Evolution): Bir kere sistem geliştirildiğinde, modifikasyon ve eklemeler gerektirecek değişikliklerin

---

[71] Savic,Düsko. **Object Oriented Programming With SmallTalk / V.** England, Ellis Horwood Limited, 1990, s. 15

[72] Booch, Grady. **Object Oriented Design.** Redwood City, Benjamin/Cummings,1991, s. 90

oluşması muhtemeldir. Bu değişikliklere kısmen dinamik ortamlarda ihtiyaç duyulacağından, tasarım esnasında bu ihtiyaçların giderilmesi oldukça zordur. Genellikle sonradan ortaya çıkan ihtiyaçların giderilebilmesinde soyutlama kavramının bu prensibi kullanılır.

2. Gelişme İle Birlikte Çözüm (Solution by Evolution): Problemlere daha genel çözümler üretmek amacıyla, ilk deneylerden yazılım üretiminin son aşamasına kadar artan tarzda çözümler geliştirmektir. Nesneye yönelik yaklaşımın felsefik bakış açısına göre, her iki görüş tek bir yaklaşım olarak kabul edilebilir. Bundan dolayı gelişme (evolution), sistemin tamamını, başlangıç adımından bakım aşamasına kadar her zaman etkileyen bir soyutlama prensibidir.

2.2.7.4. Doğrulama (Correctness)

Nesneye yönelik soyutlamanın son prensibi doğrulamadır. Doğruluk terimi program doğruluğu, uygunluk testi ve hata toleransı gibi işlemlerin kontrolünde kullanılır. Her bir nesne, sistemdeki diğer nesnelere özel (specific) davranışlarda bulunur. Diğer bir deyişle her nesne için özel davranışlar söz konusudur. Yani doğrulama prensibi, nesneye yönelik sistemlerde nesnelerin spesifik davranışlarının doğruluklarını kontrol eder.

2.2.8. Nesneye Yönelik Yaklaşımın Uygulama Alanları

Nesneye yönelik yaklaşım, birçok farklı alanda kendini kabul ettirmiş ve yaygın olarak kullanılır olmuştur. Bir çok disiplindeki araştırmacılar, nesneye yönelik yaklaşım kavramlarının avantajlarından etkilenerek bu kavramları uygulamaya çalışmaktadırlar. Nesneye yönelik kavramların etkin oldukları alanlar şu şekilde sıralanabilir:[73]

1. Programlama Dilleri: Nesneye yönelik yaklaşıma dayanan düşünceler, ilk olarak, sadece programlama dillerinin var olduğu dönemlerde ortaya çıkmıştır. O zamanlardan bugüne kadar da en etkili oldukları alan, yine programlama dilleri olmuştur. İlk dikkatler, Simula ve Smalltalk gibi dillerin üzerinde yoğunlaşmıştır. Ancak sonraları daha farklı dillerin gelişmesi ile nesneye yönelik düşüncelerin uygulanması da bu diller ile yapılmaya

---

[73] Cox, B. J. **Object-Oriented Programming: An Evolutionary Approach.** Addison-Wesley, Massachusetts, 1987, s. 46

başlanmıştır. Örneğin; Eiffel, Beta, C++, ACT-1 ve Turbo Pascal gibi diller günümüzde nesneye yönelik kavramların en çok uygulandıkları programlama dilleridir.

2. Yazılım Mühendisliği: Nesneye yönelik yaklaşım, yazılım mühendisliği alanında da büyük ilgi görmüştür. Yazılım mühendisleri, nesneye yönelik örnekleri bir tamamlama aracından, geniş bir araç ve teknikler topluluğuna dönüştürmekle ilgilenmişlerdir. Yapılan çalışmaların çoğu nesneye yönelik tasarım olarak gerçekleştirilmiştir. Bunun yanısıra, yazılım mühendisliği alanında, nesneye yönelik özel programlama dillerini kullanan bir çok proje de mevcuttur.

3. Veri Tabanları: Nesneye yönelik veritabanları, daha geleneksel ağ (network) modellerine karşı geçerli bir alternatif olarak ortaya çıkmaktadır. Bilgisayar destekli tasarım sistemleri ve yazılım mühendisliği ortamları gibi tasarım ortamlarını desteklemek için veri tabanlarında nesneye yönelik tekniklere ilişkin özel bir ilgi vardır. Ticari amaclı olarak mevcut bulunan nesneye yönelik veri tabanlarından Iris Gemstone ve Orion en çok kullanılan nesneye yönelik veri tabanlarıdır.

4. Yapay Zeka: Yapay zeka araştırmacılarının ilgisini, esas olarak nesneye yönelik yaklaşımın, karmaşık bilgilerin kavramsal olarak uygun bir tarzda temsil edilmesini sağlayabilme yeteneği çekmiştir. Nesneye dayalı bilginin temsil edildiği bir çok programlama dili, mantıksal (logic) modelleri esas alarak, yapay zeka kavramlarını uygulayabilmektedir.

5. İnsan ve Bilgisayar Arayüzü: Nesneye yönelik hesaplamada büyük ölçüde yararlanılan bir alan da insan bilgisayar ara bağlantısıdır (Interface). Temsil ve etkileşim kavramları ile nesneye yönelik düşünceler arasında bir uyum sözkonusudur. Nesneye yönelik yaklaşımlardan yararlanılan arayüz örnekleri olarak, Smalltalk, Macintosh desktopları ve ofis bilgi sistemleri sahalarında kullanılan birçok yazılım gösterilebilir.

6. İşletim Sistemleri: İşletim sistemi tasarımcıları, sistem programlarının kompleks ortamlardaki yapılanmalarını gerçekleştirmeleri için gerekli olan tekniklerle her zaman ilgilenmişlerdir. Bu tasarımcıların çoğu, nesneye yönelik

yaklaşımın yararlarından faydalanarak, sistem programlarının tasarımında bu yaklaşımı esas almışlardır.[74]

2.3. NESNEYE YÖNELİK YAZILIM GELİŞTİRME METODLARI

Yazılım geliştirme, heyecanlı ancak zor bir iştir. Yazılım geliştiren kişiler gerektiğinde ve genelde kompleks problemlerle karşılaşırlar. Yazılım geliştirmede başarılı olabilmenin ön şartlarından biri esnekliktir. Esneklik, geliştirilen projelerin, ileride oluşabilecek değişikliklere uyum gösterebilmeleridir. Esnekliğe sahip yazılımlar daha geniş sahalarda uygulanabilme fırsatı bulurlar. Esnekliğin tam olması halinde gerçek dünya ile yazılım arasında birbirlerine olan bağlılık ve benzerlik artacaktır. Bu sayede yazılımı gerçekleştiren kişilerin, geliştirecekleri yazılımlara sezgilerini yansıtmaları da mümkün olacaktır.

Nesneye yönelim, bu insan sezgilerini, kullanmayı amaçlayan bir programlama yaklaşımıdır. Nesneye yönelik yaklaşımda, yazılımda kullandığımız nesneler (objeler) gerçek dünyadaki nesneler ile bağlanırlar. Bu durum geleneksel görüşten uzak bir programlama modelidir. Geleneksel programlamada, gerçek dünya ile benzerlik ve insan sezgilerinin yazılıma yansıması istenilen seviyede değildir. Nesneye yönelimin geleneksel yaklaşımla arasındaki bu fark, bazı avantajları ve bir de istenmeyen durumu beraberinde getirmektedir. Elde edilen avantaj, insan sezgilerinin yazılıma yansıması ve hem teorik hem de pratik açıdan verimliliğin oldukça fazla artmasıdır. Nesneye yönelimin uygulanması sonucunda, eski yazılım metodları ile gerçekleştirilen projelerin yeni yaklaşıma adapte edilememesi veya yanlış adapte edilmesi ise arzu edilmemesine rağmen oluşan olumsuz bir durumdur.

Nesneye yönelik yaklaşımın gelişme metodlarının ilk kısmını, gerçekte nesneye yönelik olmayan fikirlerin nesneye yönelikmiş gibi uygulanmaya konulması oluşturur. Bu, ilk nesil (first generation) olarak adlandırılmıştır. Gerçek manada nesneye yönelik metod, ikinci nesil olan ve fusion olarak adlandırılan metoddur. Fusion, nesneye yönelik yazılım gelişimine, sistematik bir yaklaşım sağlamak amacıyla geliştirilmiştir.[75]

---

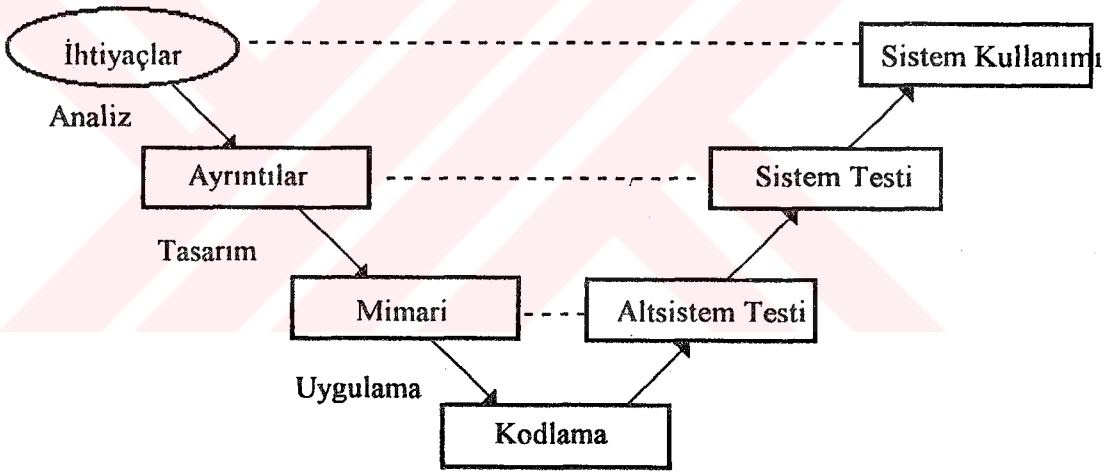
[74] Goldberg,A. **Smalltalk-80 The Interactive Programming Environment.** USA, Wesley Series in Computer Science, 1984, s.76

[75] Coleman,C. Arnold,P. ve Bodoff,S. **Object Oriented Development The Fusion Method,** USA, Prentice - Hall International, 1994, ss.1-8

Aşağıdaki kesimlerde öncelikle yazılım geliştirme işlemleri, yazılım geliştirme hatalarının kontrolü ve yazılım geliştirme problemleri ele alınmaktadır. Daha sonra da nesneye yönelik yazılım geliştirme metodu ve onunla ilgili problemler incelenmekte ve çözüm olarak da fusion metodu tanıtılmaktadır.

2.3.1. Yazılım Geliştirme İşlemi

Sistem yönetmenin, ya da diğer bir ifade ile iyi bir şekilde yazılım geliştirmeyi başarmanın yollarından biri, geliştirilecek yazılımı safha olarak adlandırılan birkaç alt bölüme ayırmaktır. Her safha kendine ait kavramlar ve semboller ile farklı problemlerin çözümü için ayrılmalıdır. Her safhadan çıkış bir diğer safha için temel oluşturmalıdır. Her ne kadar safhalar arasında sınırların tesbiti ve yerleşim şekli ile ilgili zorluklar olacaksa da safhalar doğal olarak birbirleriyle uyumlu olmalıdırlar. Şekil 2.5'de yazılım geliştirme işleminde safhaların kullanımı ve safhalar arasındaki irtibatlar gösterilmektedir.



Şekil 2.5. Yazılım Geliştirme Aşamaları

Şelik 2.5'deki elips, üretilecek sistem için orijinal ihtiyaçları gösterir. Kutucuklar ise farklı gelişim safhalarının çıktılarını oluşturur. Zaman soldan sağa doğru ilerler. Aşağı doğru yönlendirilen oklar ile safhaların akışı verilmektedir. Her bir safhanın ayrı ayrı kodlanması daha uygundur. Sağdan yukarı doğru olan oklar ile de uygulama öncesi tamamlanacak kodlamalar gösterilmektedir. Kesik çizgili hatlar ise, sağdaki hemen uygulama öncesi safha sonuçları ile soldaki başlangıç safhaları arasında ilişkilendirmeyi ve soldaki

safhalarda istenilenlere sağ tarafdaki safha sonuçları ile erişildiğini göstermektedir.[76]

Yazılım üç safhada üretilir. Her safha geliştirilmekte olan sistemin farklı görüntüleriyle ilgili faaliyetlerden oluşur. Bunları şu şekilde ifade edebiliriz.

1. Analiz: Analiz, sistemin ne yaptığının ve detaylarının ortaya çıkarılmasıdır. Amaç, konunun tam olarak kavranması ve en önemli hususların belirlenmesidir. Analiz safhasında bu tür bilgilere ait dokümanlar elde edilir.

2. Tasarım: Analiz safhasında elde edilen dokümanlar, tasarım (dizayn) safhasının kaynağını teşkil eder. Geliştirilecek sistemin amaç ve sınırlamaları dikkate alınarak, sistem elemanlarının özel davranışları tesbit edilir. Bu davranışların nasıl ve hangi yollarla sağlanacağı belirlenir ve karara bağlanır. Bu safhanın sonucunda da detaylı bir doküman elde edilir.

3. Uygulama:Tasarım safhası sonucunda, üretilen doküman dikkate alınarak, geliştirilecek projenin kodlama işlemlerine başlanır. Kodlama genelde parçalar halinde yapılır. Üretien parçalar test edilir. Daha sonra bir araya getirilen yazılımlar, sistemi oluştururlar.

2.3.1.1. Yazılım Geliştirme Hatalarının Kontrolü

Geliştirilen yazılımlarda, hata ihtimali her zaman vardır. Önemli olan hataların kabul edilebilr seviyelerde olmaları ve zamanında tesbit edilebilmesidir. Kaliteli kodlama, mümkün olduğu kadar az hata içermelidir. Bunu sağlamak amacıyla, safha bazında hata kontrolü yapılmalıdır. Hata kontrolü yaklaşımında, geliştirme ve kodlama bölümü veya montaj bölümlerinde yapılacak kontroller, temel olarak farklı yaklaşımlardır. Montaj bölümünde hata kontrolü, oluşmuş hataların bulunup çıkarılarak kaliteli bir ürün elde edilmesini amaçlayan, geliştirme veya kodlama bölümlerinde yapılacak hata kontrolleri ise hatanın hiç oluşmamasını sağlayan, hatadan kaçınma amacında olan kontrollerdir.

Deneyler göstermiştir ki hata ne kadar geç teşhis edilirse, düzeltilmesi o kadar güç ve maliyeti de o kadar yüksek olur. Bir sistem geliştirlmesi sırasında

---

[76] Blair, G. Gallagher, J. Hutchisaon, D. ve Shepherd, D. s.71

oluşan bir hata, modelin analiz işlemleri safhasında anlaşıldığında onun tamiri nisbeten ucuz olacaktır. Bununla birlikte analiz safhasında oluşan ve test işlemleri sırasında tesbit edilebilen bir hatanın telafisi çok güçtür. Potansiyel olarak gerekli bütün safhaların yeniden çalışılmasını gerektirir. Bu çoğu zaman ya imkansızdır ya da maliyeti çok yüksek olan bir işlemdir. Bu nedenle, her üretim safhasının sonunda hata kontrolü yapılmalı ve bir sonraki safhaya hatasız geçilmelidir. Güvenli kabul edebilecek bir metod, hem hata yapma ihtimali daha az olan teknikleri kullanan, hem de hataları elimine edebilen metoddur. Hatalardan kaçınmak onları düzeltmekten daha etkili olduğu için, Fusion metodun odaklaştığı safha, yazılımın uygulama işlemlerindense, üretim safhalarıdır.

2.3.1.2. Yazılım Geliştirme Problemleri

Yazılım geliştirmede oluşan problemler, 1960'dan günümüze kadar gelmektedir. Bu hatalar zaman zaman, yazılım sektöründe krizlerin doğmasına neden olmaktadır. Yazılım geliştirme sırasında karşılaşılabilecek en önemli problemleri şu şekilde sıralayabiliriz:[77]

1. Zayıf Öngörü: Yazılım geliştirecek kişiler, kullanıcıların ihtiyaçlarını tam manasıyla karşılayacak bir sistemi ne şekilde, ne kadar sürede ve ne kadar emekle geliştirebileceklerini sağlıklı olarak tahmin edemeyebilirler. Bu durum istenilen projenin kullanıcının istediği süre içinde teslim edilmesini engeller ve tahmin edilen tarihten çok sonra kullanılmasına neden olur.

2. Düşük Kalite: Projeyi kullanacak kişi veya kişilerin isteklerinin yazılımcı tarafından yeterince anlaşılamaması veya yazılım esnasında oluşabilecek veri kaybı veya yazılımın biran önce bitirilmesi yönünde gösterilecek acelecilik yazılımın kalitesini düşürebilecek unsurlardır. Bu durumda orijinal ihtiyaçlar tam gerçekleştirilemeyebilir. Düşük kaliteli program üretiminin en kötü yanı ise, bu durumun çoğu kez işlemlerin son aşamasına kadar anlaşılamamasıdır.

3. Yüksek Bakım Maliyetleri: Geliştirilen projelere daha sonra yeni özellikler eklenmek istenmesi veya oluşan bir hatanın telafisine çalışılması ya da yazılımın çoğaltılmak istenmesi durumlarında, orijinal yazılımın nesneye

---

[77] Jackson, M. A. s. 88

yönelik olmaması nedeniyle istenilenlerin yapılabilmesi, oldukça yüksek maliyetler gerektirecektir.

4. Çabaların Tekrarı: Projeler daha evvelki projeler ile uyum gösterebilir. Hatta tek bir proje içinde bile, aynı algoritmalar farklı problemler için kullanılabilir. Ama arzu edilen bu durumun, geleneksel dillerde kullanımı çok güçtür. Eski kodların yeniden kullanımı veya değiştirilerek kullanımı, geleneksel metodlarla ve geleneksel dillerle sağlanamaz. Bu durum, zaten güç olan yazılım geliştirme işleminin, benzer konularda bile her seferinde yeniden yapılmasını gerektirecektir.

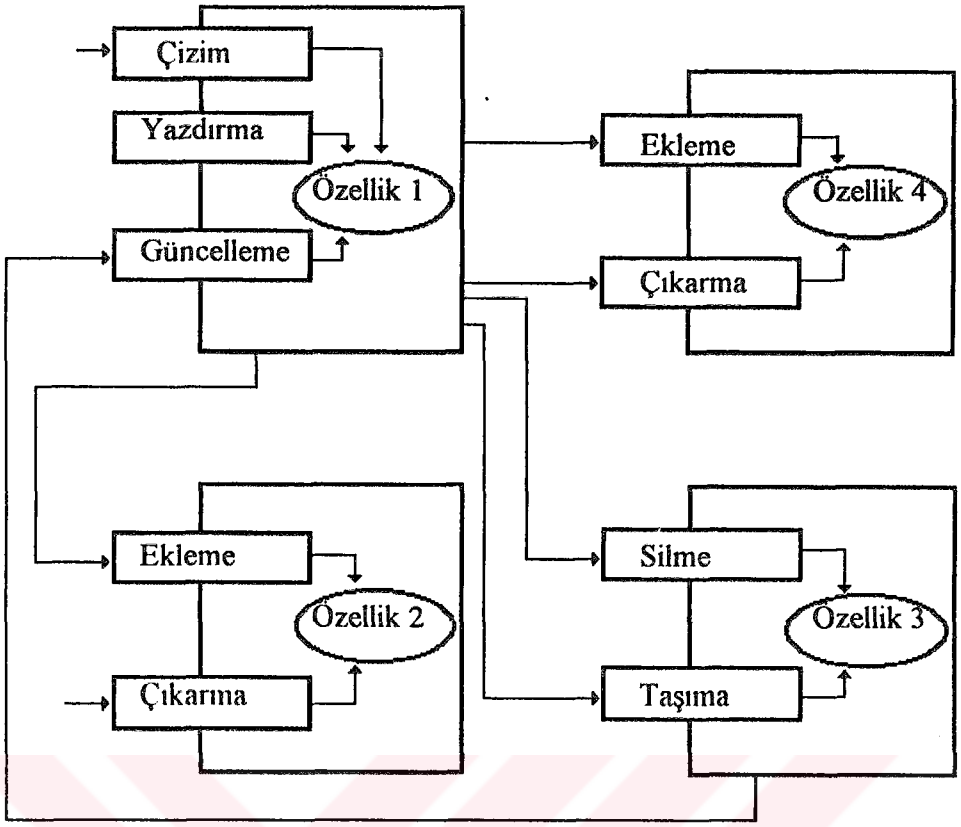
2.3.2. Yazılım Geliştirmede Nesneye Yönelik Bakış

Nesneye yönelik yaklaşımda en temel unsur, birbirlerine mesaj gönderen nesnelerdir (objelerdir). Bu mesajlar, lüzumlu hareketleri ve işlemleri icra eden metodların oluşumunu sağlarlar. Mesaj gönderen nesne, diğer nesnelerin kendi içinde nasıl düzenlendiğini bilme ihtiyacını duymaz.[78] Bu model şekil 2.6'da bir örnekle detaylı bir şekilde gösterilmiştir:

Şekil 2.6'daki dikdörtgenler, nesneleri temsil etmektedirler. Oklar ise aralarındaki mesajları gösterir. Diyagram üzerinde oval olarak gösterilen nesnelerin durumları kendi içlerinde saklıdır. Nesnenin durumunu okumanın ya da değiştirmenin tek yolu, ona mesaj göndermektir. Nesneler gruplandırılmıştır. Bir çok nesne bir sınıf altında tanımlanabilir. Sınıflar, nesneye yönelik programlama dillerinin yapı bloklarıdır. Model dinamiktir. Sistem çalıştırıldığında (execute), nesneler, işlemlerini aralarındaki mesajlarla gerçekleştirirler.

---

[78] Rambaugh, J. Michael, B. William P. ve Frederick E. **Object-Oriented Modeling and Design.** Englewood Cliffs, Prentice-Hall, 1991, s.167



Şekil 2.6. Dinamik Model

Nesneye yönelik yaklaşımla yazılım geliştirmenin avantajları şu şekilde sıralanabilir:

1. Veri Soyutlaması: Bir sınıfın farklı uygulamaları, o sınıfı kullanan kodlarda değişiklik yapılmadan sağlanabilir.

2. Esneklik: Sınıflar, yazılım gelişiminde görev dağılımı için doğal bir ünite oluştururlar.

3. Yeniden Kullanım: Sınıfların oluşturulmasında, nesnelerin temsil edilebilmeleri, yeniden yazılım geliştirme durumlarında aynı sınıfların yeniden kullanılmalarını mümkün kılar.

4. Genişleyebilirlik: Nesneye yönelik tekniklerle hazırlanan yazılımlar genişlemeye müsaittir. Bunun iki nedeni vardır: Birincisi, eski sınıflardan yeni sınıflar oluşturmak mümkündür. İkincisi ise, sınıfların farklı alanlarda kullanımı mümkündür. Bütün bunlar değişimi ve genişlemeyi kolaylaştırır.

5. Bakım: Sınıf yapısı, doğal modül oluşumunu sağlar ve yapılacak değişikliklerin etkilerini olumsuz kılmaz. Nesneye yönelimin kalıtım (inheritance) özelliği, gereksiz kod kullanımına engel olur ve yazılımın anlaşılması ve gerekli değişikliklerin yapılmasını kolaylaştırır.

2.3.2.1. Nesneye Yönelik Bakışın Problemleri

Nesneye yönelik yaklaşım tekniği, sistem gelişitirilmesinde büyük avantajlar sağlamasına rağmen, birçok problemi de beraberinde getirmektedir.[79] Bunları şu şekilde sıralayabiliriz:

1. Nesneye yönelimde genel vurgu sadece programın geliştirilmesinde değil, programlama dilleri ve programlama teknikleri üzerindedir. Programlama terimleriyle ifade edildiğinde analiz ve tasarım safhaları yeterince belirgin değillerdir.

2. Nesneye yönelik yazılım geliştirme yöntemi ile yazılım geliştirme, ekip çalışması gerektirir. Bu aslında kompleks projelerde, arzu edilen kalitede bir yazılım geliştirmenin de ön koşuludur. Ancak ekip çalışmasının gerçekleştirilmesi her zaman mümkün olmaz.

3. Nesneye yönelik bir sistemde, gerçek dünya ile uyumlu nesnelerin bulunması ve bunların doğru sınıflandırılmaları kolay değildir.

4. Geleneksel yaklaşım ile gerçekleştirilen bir yazılım üzerinde, nesneye yönelik çalışma yapılamaz. Geleneksel analiz ve dizayn metodları geçerli değildir.

2.3.3. Fusion Metodu

Fusion, nesneye yönelik yaklaşım ile bir yazılım geliştirme metodudur.[80] Fusion analiz, tasarım ve uygulama safhalarını bir arada bulunduran ancak daha çok uygulama alanında etkili olan bir metoddur. Sistemde kullanılan semboller, sistemin nesnel yapısının korunmasını mümkün kılar. Mevcut yaklaşımları genişleterek ya da birleştirerek ihtiyaçların tanımından, programlama dilinin belirlenmesine kadar bütün aşamaları kontrol eder. Fusion

---

[79] Cox, B. J. s. 65

[80] Coleman, C. Arnold, P. ve Bodoff, S. **Object Oriented Development The Fusion Method.** USA, Prentice - Hall International, 1994, s. 118

belirli bir özelliği olan, tasarım kuralları ve analiz için tanımlanmış sembol ve işaretlerdir.[81]

Fusion metodunun özellikleri şunlardır:

1. Yazılımı safhalara ayırır, böler ve her safhada ne yapılması gerektiğini gösterir. Bu nedenle programcılar nasıl program yapacaklarını bilir ve sonraki safhalara nasıl ve ne zaman geçileceğini tesbit ederler.

2. Bütün safhalar için, kapsamlı, kullanımı basit ve iyi tanımlanmış semboller sağlar.

3. Her safhada, kendine ait farklı bir teknik uygular.

4. Fusion metodu, geliştirilecek programların, farklı versiyonları arasına adapte edilebilir. Fusion metodunun parçaları veya sembolleri, gelişmiş versiyonlu uygulamalarda kullanılabilir. Yeni sistemlerin geliştirilmesi fusion metodunun kullanımını engellemez.

Fusion metodu, analiz, tasarım ve uygulama safhalarının ayrı ayrı işlem görmesini benimser ve safhaları şu şekilde değerlendirir:

1. Analiz: Analiz, sistemin amaçlanan davranışını gösterir. Sistemin modelleri üretilmiştir. Sistemin var olan nesnelerinin sınıflandırılması, bu sınıflar arasındaki ilişkiler, sistemde uygulanan işlemler ve bu işlemlerin uygunlukları bu safhada belirlenmiştir.

2. Tasarım: Tasarımcı, çalışma süresince nesnelerin hareketlerini, birbirleriyle etkileşimlerini, sistem operasyonlarının nasıl tamamlanacağını tesbit eder. İşlemler sınıflara eklenir. Tasarımcı, nesnelerin birbirleri arasındaki mesajlaşmaların ve sınıflar arasındaki ilişkilerin nasıl olacağını belirler. Ayrıca, tasarım safhasında nesnelerin birbirleriyle ilişkilerinin, sistemi nasıl etkileyeceği ve sınıfların hiyerarşik durumları da tesbit edilir.

3. Uygulama: Uygulama safhasında, uygulamanın gerçekleştirileceği özel programlama dili kullanılarak, tasarım safhasında tesbit edilen hususların,

---

[81] Coleman,C. Arnold,P. ve Bodoff,S. ss. 9-11

kodlamaları yapılır. Fusion metodu, nesneye yönelik yazılım uygulamaları için değiştirilmesi gereken test teknikleri ve geleneksel araştırmaların, nasıl değiştirilebileceği konusundaki tavsiyelerini ve geliştirilecek yazılım performansının seviyesini bu safhada belirtir.

2.3.4. Nesneye Yönelim Stratejisi

Son olarak, nesneye yönelik metodların kullanımı için yazılım geliştirilmesinde kullanılan bir stratejiden bahsedilebilir. Strateji, aşağıdaki özellikleri kapsamalıdır:

1. Problem çözümüne geçmeden önce, sistem genel olarak incelenmelidir.

2. Sistemin geliştirilme süresi belirlenmelidir.

3. Sistemin safhalara ayrılmasını sağlayan, nesneyi esas alan bölünmeler yapılmalıdır.

4. Yeniden kullanılabilirlik hedeflenmelidir.

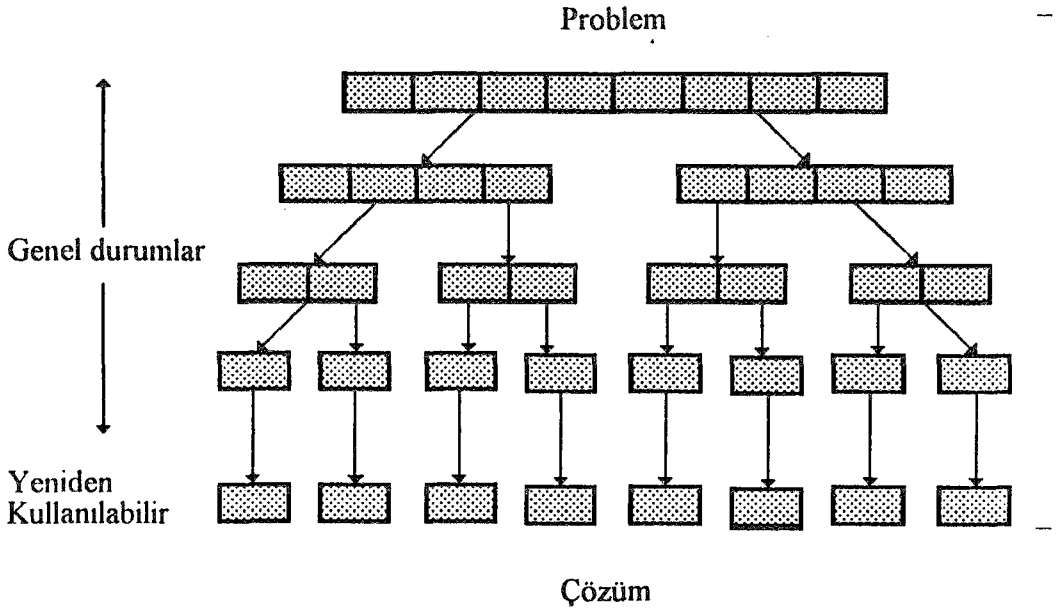
5. Yoğunlaşmaların fonksiyonlar üzerinde değil, nesneler üzerinde odaklaşması sağlanmalıdır.

6. Bölümler kendi içlerinde tam olarak bitirilmeli, diğer bölüm ile irtibatı sonra sağlanmalıdır. Ancak sistemin tümünü etkileyecek hususların, bölümler dikkate alınmaksızın gerçekleştirilmesi esas alınmalıdır.[82]

Nesneye yönelim stratejisi, şekil 2.7'de gösterilmektedir. Görüldüğü gibi, ilk aşamada sistem genel olarak incelenmiş, sonra iki bölüme ayrılmıştır. Daha sonra her bir bölüm kendi aralarında alt bölümlere ayrılmıştır. En son aşamada, üretilen yazılım parçaları kendi başlarına fonksiyonları olan ve yeniden kullanılabilir yazılımlar halini almışlardır. Ayrıca sisteme ait genel özellikler, diğer bölünmeler dikkate alınmadan, proje boyunca sürdürülmüştür.

---

[82] Florentin,J,J. **Object Oriented Programming Systems Tools and Applications.** Newyork, Chapman & Hall, 1991, s. 72



Şekik 2.7. Nesneye Yönelim Stratejisi

2.4. NESNEYE YÖNELİK TASARIM

Gelişmeye elverişli, kolay anlaşılabilir programlar üretmek, iyi bir tasarım temeli kurmaya bağlıdır. Nesneye yönelik programlamada iyi bir tasarım, sınıfların, nesnelerin ve üye fonksiyonların gerçekçi bir şekilde oluşturulmaları ile mümkün olabilir.[83]

Sınıf ve nesneler, karmaşık yazılım uygulamalarını, donanım kaynaklarını, olayları veya uygulamada gerek duyulabilecek diğer durumları temsil etmek üzere tasarlanmalıdır. Bazı alanlarda, fonksiyonların tasnif edilmesi ve veri akış analizi gibi diğer teknikler de tasarım amacıyla kullanılabilir.

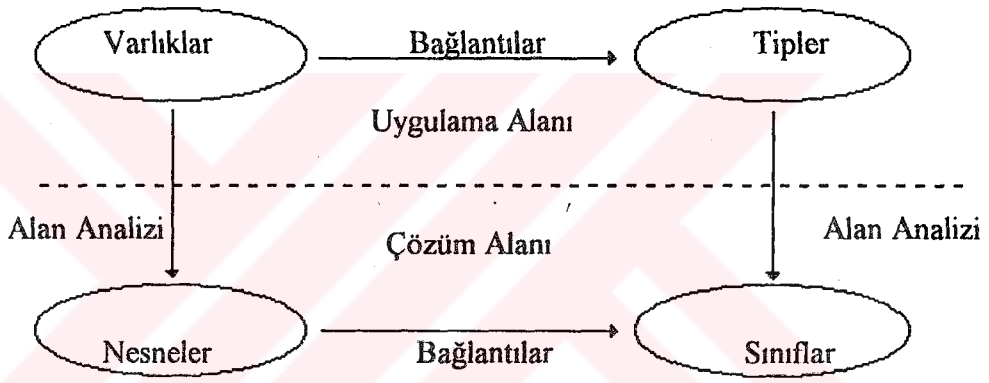
Oluşturulan sınıflar, genellikle kullanılan programlama dili için, yeni tip tanımlamalarının yapılabilmesini sağlar. Tip kavramı, önceleri belirli bir topluluğun genel özelliklerini, temsil etmek üzere küme oluşturmada kullanılmıştır. Sayısal hesaplamalar, önceleri çoğunlukla mühendislik ve matematik uygulama alanlarına yönelik olduğundan kullanılan tipler ihtiyaca cevap verebilmekte ve kullanılan dilin değişkenlerine ve tiplerine karşılık gelebilmekteydi. Fakat programlama dilleri gerçek dünyayı makina dünyasına yansıtmak için, birtakım yaklaşımlar ve sınırlamalar yapmak zorundadır.

[83] Robinson, Peter. s. 98

Örneğin sayı kümeleri, gerçek anlamda sınırsız değildir. Gösterilebilecek en küçük ve en büyük sayı değerleri, kullanılan bilgisayar donanımının sözcük uzunluğuna ve uygulanan sayı formlarına bağlıdır.

Sınıflar, tiplerin taşıdığı özellikleri taşırlar. Nesneye yönelik programlamada buna ek olarak, programcının kullanıcı isteklerine uygun tipler oluşturması, bunlar üzerinde, istediği işlemleri uygulayabilmesi ve sınıflar arasında bağlar kurması mümkündür.[84]

İyi bir nesneye yönelik tasarımda, uygulama alanında (application domain) tipler, çözüm alanındada (solution domain) sınıflar kullanılır. Uygulama alanında ortaya konan varlıklar nesnelerle, tipler de sınıflarla gösterilir. Şekil 2.8'de bu ilişkiler gösterilmektedir.



Şekil 2.8. Alanlararası İlişkiler

Alanlararası ilişkileri göstermek amacıyla, şekil 2.8'de, bir banka sisteminin işlem, para, hesap ve otomatik makinaları, bir haberleşme sistemine ait telefonlar, konuşmalar, hatlar ve anahtarlama kutuları, uygulama alanı varlıklarına örnek olarak verilmektedir. Bu varlılar genelleştirilerek, tip haline dönüştürülür ve bireylere ait varlıklar yerine, genel olarak işlemlerden, hesaplardan, telefonlardan bahsedilebilir. Uygulama alanına ait bu kavramlar, yazılımın geliştirileceği çözüm alanında tiplerin sınıflara, varlıkların da nesnelere dönüştürülmesinde kullanılır. Varlıkların veya bunların birbirleriyle ilişkilerinin, iyi belirlenmemesi halinde önemli problemler oluşabilir. Bu nedenle, uygulama alanında incelemelerin iyi yapılması, varlıkların ve bunların

[84] Birchenough, A. ve Cameron, J, R. "JSD and Object-Oriented Design". **7th ada UK Conference**. New York. 1989, s. 67

birbirleriyle olan ilişkilerinin, doğru tesbit edilmesi gerekmektedir. Nesneye yönelimde uygulama alanındaki ilişkileri belirlemede genel kalıtım (public inheritance) kullanılır. Genel kalıtım ile türetilen bir sınıf, temel sınıfın tüm özelliklerini kendisinde taşır. Fakat otomatik kalıtım şekli, temel sınıfın sadece türetilen sınıf tarafından erişilebildiği özel kalıtımdır (private inheritance).

2.4.1. Nesneye Yönelik Tasarım Aşamaları

Yazılan bir uygulamanın başarılı olabilmesi, tasarımın niteliğine bağlıdır. Nesneye yönelik tasarım, kuşkusuz, en iyi tekniklerden birisidir. Çünkü sistemi oluşturan temel parçalar, kullanıcının tanıdığı, görebildiği, kullanabildiği ve her birinin ayrı birer sınıf oluşturabildiği varlıklardır.[85] Tasarımcının, sistemi kullanacak olan kişilerle yakın ilişki içinde bulunması ve sistemin amacı ile işleyişini tam olarak bilmesi gereklidir. Kullanıcıların isteklerinin, değerlendirilmesinden ve sistemin ana hatlarının belirlenmesinden sonra, programcılar işe başlarlar. Başarılı bir nesneye yönelik tasarımın, aşamaları şu şekilde sıralanabilir:

1. Uygulama Alanındaki Varlıkların Belirlenmesi: Eğer bir kullanıcı uygulama içinde herhangi bir ismi kullanacaksa, onu varlık olarak değerlendirmek gerekir. Varlıkların belirlenmesinde kesin kurallar olmamasına rağmen, dikkat edilmesi gereken hususlar şunlardır:

Bir varlığın, adı ve görevi kolay belirlenebilmelidir. Bunu belirlemek kolay değilse, bu varlığın bir fonksiyon veya bir yapı olarak belirlenmesi daha doğrudur.

Varlıkların davranışları, kendi aralarında ilşkili olmalıdır. Eğer bir varlık, daha genel bir varlığın, özel bir şekli ise, daha sonraki aşamalarda kullanım kolaylığı sağlamak amacıyla, bu ilişki belirtilmelidir.

Varlıkların kapsamları, ne çok büyük, ne de çok küçük olmalıdır. Uygun olduğu düşünülen büyüklükler için, küçük varlıklar birleştirilmeli, büyük varlıkların ise kapsamları daraltılmalıdır.

---

85 Heitz, M. ve Labreville, B. "Designand Development of Distributed Software Using Hierarchical Object-Oriented Design and Ada". **Ada-Europe International Conference.** Munich. 1988, s. 35

2. Varlıkların Davranışlarının Belirlenmesi: Uygulama için ihtiyaç duyulan tüm görevler belirlenmeli ve bu görevleri yerine getirebilecek varlıklar, tesbit edilmelidir. Varlıkların davranışlarının, açık, anlaşılır, kısa ve öz olmaları istenilen bir durumdur. Varlıklar, üye fonksiyon olarak düşünüldüklerinde, genellikle bir veya iki parametre almalıdırlar.

3. Test: Bu aşamada, sistem, bir bütün olarak test edilmelidir. Kullanıcının isteklerine uygunluğu, başarı seviyesinin yeterliliği, kabul edilebilir veya ihmal edilebilir hata seviyesine inilmiş olduğu gözlenmeli ve tüm seçenekler denenmelidir.

Nesneye yönelik tasarım iki farklı yöntemle gerçekleştirilebilir. Bunlar statik sistem tasarımı ve dinamik sistem tasarımı şeklinde belirtilmektedir.

2.4.2. Statik Sistem Tasarımı

Statik sistem yapısı, sınıf diyagramları ile oluşturulur. Bu diyagramlarda çeşitli işaretleme yöntemleri ile, nesneler arası ilişkiler gösterilir. Projeler için, her aşamada bu diyagramların hazırlanması, kullanılan sınıfları birarada görebilmek açısından oldukça faydalıdır. Nesnelerle yapılan tasarımda, nesneler arasında bulunan hiyerarşi önem kazanmaktadır. Ana sınıf ve türetilmiş sınıfların statülerinin iyi tesbit edilmesi gerekir. Ayrıca, sisteme yapılabilecek yeni yazılım ilaveleri, büyük değişiklikler gerektirebilir. Büyük sistemlerde düşünülen küçük bir değişiklik, büyük problemler çıkarabilir. Statik sistem tasarımında kullanılan araçların başlıcaları, işlem diyagramları ve çatılardır. Bunlara ilaveten yazılım modülleri, alt sistemler ve kütüphaneler de statik tasarım amacıyla kullanılan araçlardır.

2.4.2.1. İşlem Diyagramları

Nesneler, üye fonksiyonlar aracılığı ile birbirleriyle haberleşirler ve birbirleri üzerinde işlem yapabilirler. Karşılıklı ilişkiler nedeniyle, bir işlemi gerçekleştiren üye fonksiyonların hangi nesnelere ait olacakları, özellikle büyük sistemlerde karmaşıklık yaratabilir. Nesneye yönelik programlamanın temelinde ise, nesnelerle işlem yapmak vardır. Bilhassa karmaşık sistemlerde, nesneye yönelik tekniklerden vazgeçilmesi halinde, tasarımın etkinliği azalacaktır. Karşılıklı fonksiyon çağırmalarında veri yapılarına erişimin serbest olması, bilgilerin yanlış kullanımına neden olabilir. Bu durum ise programın

güvenilirliğini azaltır, yazılımın anlaşılmasını güçleştirir ve geliştirilmeyi engeller. Oysa nesnelere bağımlı yaklaşım, her işlemle ilgili bilgileri uygun şekilde dağıtır.[86]

Tasarım sırasında kullanılan araçlardan biri olan, işlem diyagramları (processing diagram), nesneleri ve bunlar üzerindeki uygulamaya ilişkin işlemleri belirlemeye yardım eder. Oldukça kullanışlı olan bu diyagramlar bir tablo şeklinde olup, her sütun bir işlemi, her satır da bir nesneyi gösterir. Satır ve sütunların sıralanışı önem taşır. Mantıksal olarak birbiriyle ilşkili fonksiyonlar, tabloda yanyana yer almalıdır. Tablonun sol kenarı sağ kenarına, üst kenarı da alt kenarına bağlantılı olarak düşünülür. Tablo, soldan sağa doğru dolaşıldığında, sistem çalışırken sırayla yapılması gereken işlemler ortaya çıkar. Şekil 2.9'da, bir uçağın uçuş kontrol sistemini oluşturan tasarım gösterilmektedir:[87]

	Kalkış	Tırmanış	Düz Uçuş	Sağa Dön	Sola Dön	Alçal	İniş
Flaplar	Kadır						İndir
Sol Kanatçık							
Sağ Kanatçık							
Sol İrtifa Dümeni		Kaldır		Kaldır	Kaldır	İndir	
Sağ İrtifa Dümeni		Kaldır		Kaldır	Kaldır	İndir	
Dümen							
Gaz	Artır	Artır		Artır	Artır		

Şekil 2.9. Uçuş Kontrol Sistemleri İçin İşlem Diyagramları

86 Locus, H. C. s. 128
87 Booch, Grady. s.176

Birbirleriyle ilişkili olan nesne yapıları, ortak bir temel sınıf etrafında birleştirilebilir. Örneğin, uçağın kuyruğunda bulunan kanatçıklar, yani irtifa dümenleri, kalkış sırasında yukarı kaldırılır. Bu nedenle her iki kanatçık da aynı fonksiyona sahiptir. Burada bir İrtifa Dümeni sınıfı, temel sınıf görevi görür ve Tırmanma isimli bir üye fonksiyonu bulunur. Bu sınıftan, Sağ İrtifa Dümeni ve Sol İrtifa Dümeni sınıfları türetilebilir. Burada dikkat edilirse, sağ ve sol dümenlerin birbirlerinden ayrı herhangi bir fonksiyona sahip olmadıkları görülür. Dolayısıyla, bu iki satır birleştirilebilir ve tek bir İrtifa Dümeni nesnesi oluşturulabilir. Bunun yanısıra, hiç bir grup içine alınamayan üye fonksiyonlar da bulunabilir. Kalkış sırasında, pervanenin dönüş yönüne çekişini karşılamak için dümeni sağa kıran, Sağa Dön işlemi bunlardan biridir. Böyle işlemleri diğerleriyle birleştirmek yerine doğrudan yazmak daha doğrudur.

2.4.2.2. Modüller

Modül, sistemi oluşturan birçok birimden birisidir. Nesneye yönelime uygun programlama dillerinde, program kodunun ayrı dosyalarda tutulduğu, bir modül sistemi desteklenmektedir. Bir modül, bir veya daha fazla sınıfı içerir. Başka tiplerin veya sabitlerin tanımlanarak, dışarıdan kullanılabilmelerini sağlar. Bir modülde, iki kesim vardır: Birincisi, dışarıdan diğer sınıflar veya fonksiyonlar tarafından erişilmesine müsaade edilen sembollerin bulunduğu kesim. Bu kısma başlık dosyası denir. Başlık dosyası içinde genellikle tip tanımlamaları, ikinci kesimde (kaynak dosya) bulunan sınıfların ve global değişkenlerin bildirimleri, global fonksiyon prototipleri, derleyici direktifleri ve tanımlanan sabitler bulunur.

İkinci kesim ise, özel (private) kısımdır. Bu kesim ise, kaynak dosyası ya da Beden dosyası şeklinde isimlendirilir. Kaynak dosyasında, o modüle ait sınıfların üye fonksiyonlarının, global fonksiyonların, global nesnelerin ve değişkenlerin tanımları bulunur. Modülde, kaynak dosyanın başında hangi başlık dosyalarının kullanılacağı belirtilmelidir.

2.4.2.3. Altsistemler

Bir altsistem, bağımsız olarak taşınabilen bir modüller topluluğudur. Aynı alandaki amaçları gerçekleştirmek üzere, sıkça kullanılan modülleri

birleştirerek, altsistemler oluşturmak mümkündür.[88] Bir kez üretilen bir altsistem, başka uygulamalarda da kullanılabilir.

Altsistemlerin kullanımında, belirli bir kural yoktur. Sistemin herhangi bir düzeyinde kullanılan bir altsistem, bir başka düzeyde nesne olarak da kullanılabilir. Kullanımın nasıl olacağı, yani nesne olarak mı, altsistem olarak mı kullanılacağı, o sistemin ve uygulama alanının tümü gözönüne alınarak belirlenir.

2.4.2.4. Çatılar

Bir çatı (framework), kullanıcı isteğine göre tekrar düzenlenebilen ve geliştirilebilen bir altsistemdir. Bunun için kullanılan altsistem tasarımı, sınıflarla yapılır. Soyut temel sınıflar, bazı fonksiyonların tanımlanmasını, uygulama zamanına bırakır.

Çatı oluşturulurken kullanılacak yöntem, temel sınıfın üye fonksiyonlarını kullanıma sunup, ayrıntıları özel kısımlarda saklayarak, kullanıcının, sonradan kendi ihtiyacına göre tanımlayabileceği sanal fonksiyonları oluşturmasını sağlayacak şekilde olmalıdır.

Sonuçta az sayıda genel fonsiyonlara sahip, temel sınıflardan oluşan, bir çatı ortaya çıkar. Böyle bir çatıyı, kütüphane olarak hazırlamak da mümkündür.

2.4.2.5. Kütüphaneler

Bir kütüphane (library), programcının uygulamadaki ihtiyacına göre seçip, kullanabileceği, yazılım parçalarının topluluğudur.[89] Bir altsisteme benzer, fakat normal bir yazılım organizasyonunda, daha geniş düşünülmesi gereken, taşınabilir, bir tasarım aracıdır.

Kütüphaneler genellikle iki bölüm halinde kullanılırlar.

1. Nesne Kodlarının Bulunduğu Kısım: Bu kısım, bağdaştırma sırasında ana programa dahil edilir.

---

[88] Dittrich, K. R. **Object-Oriented Database System: The notions and the Issues.** New York, Computer Science Press, 1986, s. 61

[89] Keyes, Jessica. "Languages: The New Generation". **Computerworld.** New York, McGraw-Hill, 1990, s. 50

2. Başlık Dosyaları: Bu dosyalarda, kütüphane içinde bulunan sınıf fonksiyon ve sabitlerin bildirimleri mevcuttur.

Paketleme o şekilde yapılır ki, kütüphanenin az bir kısmı kullanılsa dahi, kullanılmayan kısım çalışan programa ek bir yük getirmez. Her kütüphane belli bir konu dikkate alınarak oluşturulmalıdır. Yazılımda kullanılan kütüphanelere standart Turbo Pascal kütüphanesini, grafik ve matematik kütüphanelerini örnek verebiliriz.

Bir kütüphaneyi, tıpkı bir alt sistem gibi ayrı olarak kullanmak ve farklı projeler için pazarlamak mümkündür.

2.4.3. Dinamik Sistem Tasarımı

Dinamik sistem tasarımı, az kullanılan bir tekniktir. Çünkü dinamik sistem tasarımı, bir sistemin, bir önceki halinden geri besleme gerektirir. Dinamik sistem tasarımında, çalışma anında oluşturulan nesnelerin, birbirleri ile bağlantısı, sistemin gerçek zaman (real time) ihtiyaçlarını karşılamak için iş düzeni ve çok görevli (multitasking) çalışma özellikleri ile sağlanır.

Tüm bunlar, tipik bir işletim sistemini ilgilendiren konular olmasına rağmen, nesneye yönelim prensipleri içinde de yer alırlar. Bir sistemin dinamik yapısı, zaman içinde değişen özelliklere uyum sağlayacak şekilde tasarlanmalıdır. Bunu sağlayacak özellikler aşağıdaki kesimlerde ele alınmıştır:

2.4.3.1. İş Sıralama

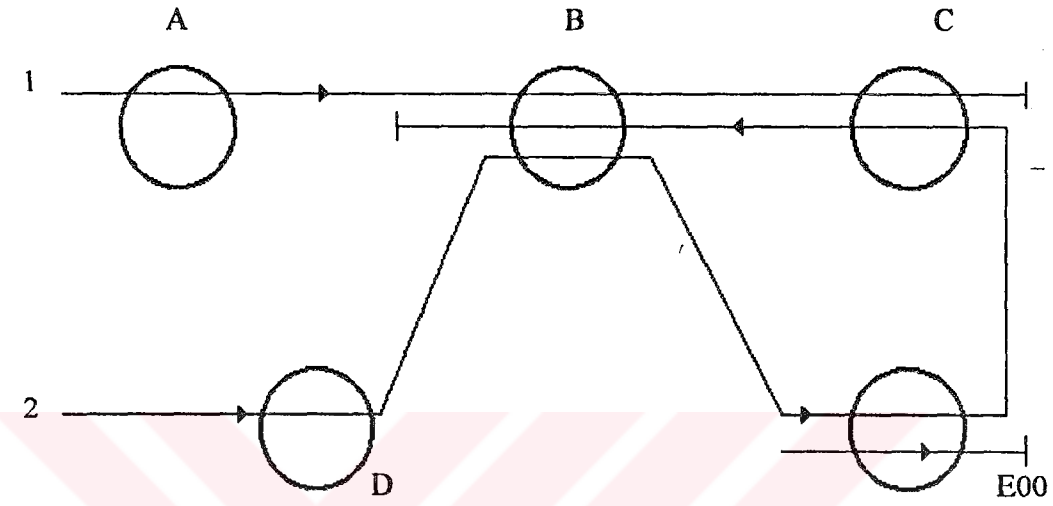
İş sıralama (scheduling), tasarımda dikkate alınması gereken önemli bir noktadır. Sıralamada yer alacak aşamaları, birer yazılım birimi gibi düşünmek mümkündür. Bu durumda programlar, belirli bir sıralamaya sokulan yapılardır. Çokişlilik (multiprocessing) ise, aşamaları birer sıralama birimi olarak kullanır.[90] Çok yaygın kullanılan bir teknik olan, çokişlilik, aşamaların, tek işlemci üzerinde ayrı bağlamlarda (context) ve sanki herbiri ayrı işlemci üzerinde çalışıyormuş gibi yürütülmesi esasına dayanır. Nesneye yönelik programlarda bir aşama, nesne olarak tasarlanıp kullanılabilir. Bunun tersi olarak, her nesnenin bir aşama olarak değerlendirilmesi ise her zaman için geçerli değildir.

---

[90] Sarıdoğan,M,Erhan. ss. 297-319

2.4.3.2. Bağlar

Bağ (thread), kendi program sayacı, yığını ve bağlantısı olan bir alt süreçtir. Bağlar, nesneler için, doğal bir sıralama sağlarlar. Bağlar, nesneye yönelik teknikleri kullanan gerçek zaman sistemleri için, önemli özelliklere sahiptir. Şekil 2.10, bağ sıralamasının nasıl yapıldığını göstermektedir.



Şekil 2.10. Nesnelerle Bağ Sıralama

Bağlar, birer istek (request) mesajı ile başlatılır. Tüm istekler, bir dağıtıcı (dispatcher) tarafından dağıtılır. Bu dağıtıcı diğer nesnelerin üye fonksiyonlarını kullanır. Bir numaralı isteğin dışarıda üretilmiş bir mesaj olduğunu varsayalım. Dağıtıcı, istek mesajını, A nesnesinin üye fonksiyonuna göndererek, onun çalışmaya başlamasını sağlar. Bu üye fonksiyon, daha sonra C nesnesinin üye fonksiyonunu çağıracak olan B nesnesinin ait üye fonksiyonu çağırmaktadır. C'nin ve daha sonra da B'nin fonksiyonları geri döndükten sonra, kontrol A'ya geçer ve bağ sona erer. Bir bağ çalışırken, dağıtıcaya başka istekler de gönderilebilir. Dağıtıcı, bağın sonunda kontrolü tekrar kazandığında, bu istek için yeni bir bağ başlatılır. Şekil 2.10'daki örnekte, bir numaralı istek, daha sonra bağlanacak olan iki numaralı isteği oluşturmaktadır.

Bağlar, olaylara karşılık verme zamanının kritik olduğu gerçek zaman uygulamalarında, yaygın bir şekilde kullanılır. Örneğin, bir veri tabanını güncelleme işlemi yapan bir bağın çalışması sırasında, tüm yazma ve okuma işlemleri bekletilmelidir.

2.4.3.3. Optimal Erişim

Sembolik programlama dillerinin, yaygın özelliklerinden biri de programcıları, bellekle uğraşmaktan kurtarmaktır. Erişilemeyen nesneler, işletim sistemi ve donanım yardımıyla, tekrar kullanılabilir duruma getirilir. Bu işlem, optimal erişimin oluşumunu sağlar. Optimal erişim ile, programcıya gerekli olan nesnelere erişim, gerekli olmayanlara da erişmeme imkanı sağlamaktadır. Sistem, bir nesnenin, başka bir nesne ya da değişken tarafından, erişiminin mümkün olmadığını sezebilirse, bu nesnenin bulunduğu alanı, başka bellek isteklerinde kullanılmak üzere serbest bırakır. Bu işlem kullanıcı tarafından farkedilmez. Bellek kullanımı, son derece önemli olduğundan, kullanılmayan nesnelerin, uygun bir şekilde yok edilebilmesi, nesneye yönelik tasarımda ele alınması gereken önemli bir konudur.

2.5. NESNEYE YÖNELİK YAZILIM GELİŞTİRME

Nesneye yönelik yaklaşım, bilgisayar yazılımlarının tasarımı ve uygulanmasında, her geçen gün daha popüler olarak kullanılan bir kavramdır. Bunun nedeni, nesneye yönelik yaklaşımın temel kavramlarından olan nesne ifadesinin, gerçek dünya nesneleri ve birbirleriyle olan ilişkilerinin, doğrudan bilgisayarda gösterilmesinde, doğal olarak kullanılmasıdır.[91]

2.5.1. Niçin Nesneye Yönelik Programlama ?

Günümüzde, en fazla rağbet gören ve önemli bir programlama yöntemi olan, nesneye yönelik programlama mantığı, özellikle çok karmaşık problemlerin çözümlenmesinde birçok kolaylıklar sağlamaktadır. Modüler olması ve adından da anlaşılabileceği gibi nesneye dayalı olması nedenleriyle, ayrıca sınıflandırma özelliği sayesinde kendini kabul ettirmiştir. Nesnelerin, birbirleriyle iletişim içerisinde olamları ve birbirlerine özelliklerini devredebilmeleri gibi özellikler bu yönteme oldukça önemli avantajlar sağlamıştır.[92]

---

[91] Güven,Doğan. **Bilişim Dizgilerinin Çözümleme Tasarım ve Gerçekleştirimi için Görsel Nesneye Dayalı bir CASE ortamı.** İzmir, Basılmamış Doktora Tezi, 1991, s. 59

[92] Demurlian,S,A. Beshers,G,M. ve Ting,T,C. "Programming Versus Databases in TheObject-Oriented Paradigm". **Information and Software Technology.** No:2. USA, Butterworth Ltd, 1993, s. 78

Nesneye yönelik programlamanın üstünlüğünü sağlayan, diğer özellikler de şunlardır:

1. Modüler yapısı ile, program tasarımının kolay olması ve çok programcı ile, modülleri farklı programcılar tarafından gerçekleştirilebilmesi, ayrıca grup çalışmasının mümkün olması.

2. Tanımlanmış olan nesne ve fonksiyonlardan, yeni nesneler türetilmesi sağlanarak, yeni alt sınıflar oluşturulmasına ve türetilen yeni nesnelerin diğer yazılımların geliştirilip kullanılmasına imkan vermesi.

3. Sağladığı esneklik ile, programlarda verimliliğin artırılması.

Bu yöntem ile program yazılırken, çözümü aranan problemin gerçek hayattaki somut veya soyut karşılıkları esas alınır. Örneğin, bu varlıklar, bir veri tabanındaki kişisel bilgiler, personel sicil bilgileri veya öğreci not bilgileri olabileceği gibi, bir grafik programındaki daire, dikdörtgen veya bir çizgi de olabilir.

Nesneye yönelik programlarda, nesneler, günlük yaşantımızda kullandığımız şekilde tanımlanır. Nesneler, özelliklerine ve davranışlarına göre gruplara ayrılırlar. Bu gruplandırma şekli, bu yöntemde sınıflama (class) özelliği ile gerçekleştirilir. Bu sayede programın modüler hale getirilmesi, bir çok programcının aynı programda farklı görevler alarak yazılım üretebilmesi mümkün olabilir. Ayrıca, yine sınıflandırma özelliği sayesinde, birbirleriyle ilişkisi olan varlıkların, ayrı ayrı tanımlanmalarına gerek olmadan, kullanılabilmeleri mümkün olmaktadır. Bir nesnenin, genel özelliklerini taşıyan bir sınıftan, alt sınıflar türetilebilir.[93] Türetilen bu alt sınıflara, sadece ana sınıftan farklı olan özelliklerinin yazılması yeterlidir.

Yazılım geliştirmek amacıyla oluşturulan algoritmalarda, her sınıfın davranışları ayrı ayrı belirlenir. Bu davranışları belirleyen fonksiyonlar, birbirleriyle mesajlaşma, yani değerlerini birbirlerine aktarma imkanlarına sahiptirler.

---

[93] Dittrich, K. R. s. 87

Bütün bunların sonucunda, oluşturulan yazılım ile, gerçek hayattaki nesnelerin davranışlarının, bilgisayarda şekillendirilmesi sağlanmış olur.

Nesneye yönelik programlama yöntemi ile geliştirilen projeler, yukarıda bahsedilen nedenlerden dolayı, şu özelliklere sahiptirler:

1. Soyutluk: Soyutluk, nesneye yönelik programlama ile geliştirilen projelerde, aynı şeyin birden fazla ifade edilmesini önlenmektir. Tekrarlama yapan çevrimler, söz konusu değildir. Nesnelerin ve sınıfların kullanılıyor olması, tekrarlanması gereken kısımların, bir modulde toplanmasını sağlamakta ve hatalardan temizlenmiş kütüphanelerin kullanılmasına imkan vermektedir.

2. Otomatiklik: Nesneye yönelik programlama, hareket özelliği olan bir yapıya sahiptir. Alt programların, prosedürlerin veya döngülerin denetimi ve icrası, bu yöntemle otomatik olarak sağlanmaktadır. Bunların sağlanması için, ayrı işlemlere gerek yoktur.

3. Güvenilirlik: Nesneye yönelik programlamada, sınıfların özel ve korunmuş kısımlarının varlığı, nesne ve fonksiyonlara erişilmesinde, güvenliği sağlamaktadır. Bu durum, verilere sadece yetkili kişilerin veya programların erişebilmeleri anlamındadır.

4. Çok Seviyeli Korunabilirlik: Bu özellik ile, her aşamada oluşabilecek kullanıcı veya donanım hataları, kontrol edilebilmektedir. Bu da, kullanımda tercih nedeni olmaktadır.

5. Taşınabilirlik: Nesneye yönelik programlama dilleri ile gercekleştirilen projelerin, EXE haline getirilmeleri mümkün olup, gerekli asgari donanıma sahip bütün bilgisayarlarda çalıştırılması mümkündür.

6. Basitlik: Nesneye yönelik programlama yöntemi ile gerçekleştirilen yazılımlarda, sınıfların, prosedürlerin ve unitlerin kullanılması, yazılımın anlaşılırlığını ve yeniden düzenlenmesini kolaylaştırmakta ve basitlik sağlamaktadır. Ancak, ilk oluşturulma aşamaşı, çok basit değildir. Bu nedenle, yeniden kullanılabilecek ve çok fonksiyonlu işlemlerde bu yöntemin kullanılması daha mantıklıdır.

7. Ucuzluk: Bu yöntemin, ileride yeniden kullanılabilmesi mümkün olan ve çok fonksiyonlu projelerde kullanılması halinde, maliyeti çok düşük olacaktır. Çünkü, hazırlanan ünitler, bir başka projede veya o projenin yeni düzenlenmiş halinde hiç bir değişikliğe uğramadan, yeniden kullanılabilmektedir. Ancak, basit ve yenilenmesi mümkün olmayan konularda, bu programlama yönteminin kullanılması, maliyet açısından da uygun değildir.

Bütün bu nedenler, nesneye yönelik programlamanın yazılım geliştirilmesinde tercih edilmesini sağlamaktadır.

2.5.2. Nesneye Yönelik Programlama Dillerinin Gelişimi

Nesneye yönelik herhangi bir programlama dili, klasik bir programlama diline bazı yeni özellik ve yapıların eklenmesi ile elde edilebileceği gibi, tamamen yeni geliştirilmiş bir dil de olabilir.[94] Nesneye yönelik programlama dilleri, yazılım sektöründe önemli bir gelişme olarak kabul edilmektedir ve bu sayede donanım sektöründe sağlanan hızlı gelişmenin, gerisinde kalınmamıştır.

Nesne kökenli programlama, kendine has özellikleri ve kavramları olan, nesneyi esas alan, nesneler arasında mesajlaşmaların mümkün olduğu ve üretilen yazılım kodlarının yeniden kullanılabildiği, kapsamlı çalışmalar için en uygun, grup çalışmasına imkan sağlayan bir programlama tekniğidir. Bu durum nesneler ile onlar üzerindeki fonksiyonların ve birbirleri arasındaki ilişkilerin önemini artırmakta ve nesneye yönelik programlama dillerini, klasik programlama dillerinden ayırmaktadır.

Nesneye yönelik programlama dilleri ile gerçekleştirilen projelerde, yazılımın tekrar başka bir proje parçası olarak kullanılabilmesi, veya diğer projelerle birleştirilebilmesi mümkün olmaktadır.

Günümüzde en çok kullanılan nesneye yönelik programlama dilleri, C++ ve Turbo Pascal'dır. Bunun yanısıra, Smalltalk ve Ada dilleri de, nesneye yönelik dillerdendir.[95]

---

[94] Margaret,M.B. ve Allen,L,A. "Declarative Visual Languages". **Journal Of Visual Languages and Computing.** No:5. USA, Academic Press Limited, 1994, ss. 1-3

[95] Kut,R,Alp. **Bir Veri Tabanı Yönetim Sistemi İçin Geliştirilmiş Nesne Kökenli Kullanıcı Arayüzü.** İzmir, Basılmamış Doktora Tezi, 1991, ss. 38-40

2.5.2.1. Smalltalk-80 Nesneye Yönelik Programlama Dili

Smalltalk 80, Xerox araştırma grubu tarafından geliştirilmiş bir programlama dilidir. Bu programlama dilinde, özel bir veri grubu ile bir veriye erişebilen fonksiyonlar topluluğu, bir nesneyi oluşturur. Buradaki fonksiyonlara, tüm kullanıcılar mesajlar sayesinde erişebilirler ve bu verileri kullanabilirler.[96] Bir mesaj ile, nesneye ne yapması gerektiği bildirilir. Nesne, kendisine ait modüllerden gerekli olanını tesbit ederek, gelen mesajlara karşılık verir. Nesnenin kendisine ait bu fonksiyon gruplarından herbirine, nesnenin metodları denir. Nesneyi uyaran, mesajı taşıyan komuta da, seçici ya da selektör denir.[97]

Smalltalk 80'de veri türlerini tanımlamak için, diğer nesneye yönelik dillerde bulunan tür tanımları ve ifade kuralları bulunmaz. Çünkü kullanılan verilerin tek bir tipi vardır. Yani her verinin tipi, nesnedir. Bu dilde ifade kuralları, sadece şu işlemler için gereklidir:

1. Nesne adlarının ve bunlara karşılık gelen verilerin atanması
2. Mesaj gönderilmesi
3. Yeni sınıf veya sınıflara ilişkin, metodların tanımlanması
4. Kullanılan komutların yazılması

2.5.2.2. Ada Nesneye Yönelik Programlama Dili

Ada, aslında Fortran, normal C ve Pascal gibi dillerden çok farklılık göstermemesine rağmen, nesne kökenli dillerin bazı özeliklerini içerir. Smalltalk'un aksine, Ada dilinde kullanılacak her değişkenin tanımlanması, programlamanın başında yapılmalıdır.[98]

Ada programlama dilinde, veri grupları ile fonksiyonlar arasındaki mesajlaşma, programın derlenmesi sırasında gerçekleştirilmektedir. Yani erken bağdaştırma (dinamik binding) yöntemi uygulanmaktadır. Bu haliyle Ada, nesneye yönelik programlamada pek de arzu edilmeyen bir yöntem kullanmaktadır. Bu nedenle de, Ada dili fazla yaygın değildir.

---

96 Tello,Ernest,R. **Object Oriented Programming for Artificial Intelligence.** Newyork, Wesley Publishing Company, 1989, s. 63
97 Florentin,J,J. ss.80-81
98 Smedema, C. H. s. 188

2.5.2.3. C++ Nesneye Yönelik Programlama Dili

C++'ın tarihçesi 1970'lerde, C'nin icadı ile başlar. C dili ilk olarak, Unix işletim sistemi ile kullanılmak üzere, Dennis Ritchie tarafından oluşturuldu. C, oldukça yaygın ve güçlü bir dildir. Son yıllarda birçok yazılım, bu dil ile geliştirilmiştir. Fakat, proje kapsamları büyüdükçe, C dili yetersiz kalmaya başlamıştır. Bu nedenle, yani kapsamlı ve daha kompleks projelere ihtiyaç duyulması nedeniyle, Amerika Birleşik Devletlerinde, Bell laboratuvarlarında çalışan Bjarme Stroustrup, programcıların kolayca, iyi programlar yapabilmeleri için, C diline çeşitli ilaveler yapmış ve bu yeni programlama diline 'C++ with classes' adını vermiştir. Bu isim, 1983 yılında Pick Mascitti'nin önerisiyle C++ olarak, değitirilmiştir. C++'a nesneye yönelik programlamayı desteklemesi için, gereken ilavelerin çoğu, Bjarme Stroustrup tarafından yapılmıştır. İlaveler yapılırken, sınıf özelliği ön plana çıkarılmıştır.[99]

C++, icadından sonra 1985 ve 1989 yıllarında, iki kere revizyondan geçmiştir. 1991 yılında, Bell laboratuvarlarında bu dilin 2.0 uyarlamasına, bazı eklemeler yapılarak 3.0 uyarlaması meydana getirilmiştir. Halen bu uyarlama yürürlüktedir.

Bu dil ile getirilen yenilik ve eklemeler kısaca şu şekilde özetlenebilir:

1. Genel Eklemeler: Yeni tip açıklama satırı, satır içi fonksiyonlar, referans tipler, fonksiyon ek kullanımı, operatör ek kullanımı öndeğerli (default) parametreler.

2. Tip Sistemine Yapılan Eklemeler: Tip çevirmeleri, fonksiyon prototipleri, kütüphaneleri, tip kontrollü bağlama ve bellek kullanım operatörleri.

3. Nesneye Yönelik Eklemeler: Sınıflar, türetilmiş sınıflar, girişim kontrolü, sanal fonksiyonlar, yardımcı fonksiyonlar, yapıcılar, yokediciler.

2.5.2.4. Turbo Pascal Nesneye Yönelik Programlama Dili

Pascal, 1968 yılında, Niklays Wirth tarafından geliştirilmiş, bir programlama dilidir. Pascal programlama dilinin özellikleri, COBOL, FORTRAN

---

[99] Ghezzi, C. ve Jazayeri, M. s.130

ve ALGOL gibi dillerden alınmıştır. Pascal, modüler programlamaya uygun çağdaş bir dildir.

Turbo Pascal ise, Borland Inc. firması tarafından geliştirilmiş, bir Pascal dili veya derleyicisidir. Turbo Pascal, standart Pascal dilinin güçlü taraflarını, güncelleştirerek benimsemiş, giriş/çıkış işlemleri gibi zayıf taraflarını ise, yeniden düzenlemiştir. Böylece mikrobilgisayarlarda kullanılan Pascal için, bir standart oluşmuştur.[100]

Yapısal programlama kavramının, ortaya atılmasıyla, programların geliştirilmesinde standart denetim yapıları kullanılmaya başlanmıştır ve geleneksel olarak kullanılan GOTO komutu, daha az kullanılır olmuştur.

GOTO komutu, bilindiği gibi, programın denetimini ileride veya geride istenilen bir yere saptırır. Bu durum programda karışıklığa yol açmakta ve GOTO komutunun kullanıldığı programların düzeltilmesi çok güç olmaktadır. Turbo Pascal GOTO'suz (goto-Less) bir dildir. Standart kontrol yapıları, okunabilirliği ve modülerliği ile kendini kabul ettirmiştir. Turbo Pascal, güçlü veri yapılarına sahiptir ve veri tiplerine çok bağımlıdır. Ancak belli koşullarla, tip dönüşümüne müsade eder.

Turbo Pascal, modüler programlama mantığına göre tasarlandığından ve unit dediğimiz, derlenmiş, kulanıma hazır, bağımsız birimleri kullanabildiğinden, değişime açık, taşınabilir, anlaşılması basit bir dildir.[101]

Turbo Pascal programlama dili ile nesneye yönelik programlama, Turbo Pascaldaki nesne özelliklerinden faydalanılarak yapılan, programlama yönteminin, genel adıdır. Nesne (object), "type" alanında tanımlanabilir bir tiptir. Nesne, hem kod (code) hem de veri (data) içerebilir. Yapısı sayesinde, sadece kayıt alanı olarak da, kullanılabilir. Fakat asıl kullanım amacı bu değildir. Şart olmamakla birlikte, nesnelerin içinde, verilerin ilk değerlerini ayarlayan, bir başlatıcı (initalize) ve nesne ile ilgili son işlemleri yapmak için, bir sonlandırıcı (done) prosedür bulunur. Tablo 2.10'da basit bir nesne tanımı verilmektedir. Burada nesne esas alınarak, unitlerin oluşturulması ve ana programda kullanımları örnek bir Turbo Pascal programı ile gösterilmektedir.

---

100 Borland, D. **Pascal:The Language and its Implementation.** New York, Wiley, 1981, s. 97
101 Çubukçu,Faruk. ss. 51-52

```
Type
Medenihal=Boolean;
Kisi=Object
        isim:string[30];  soyad:string[20];  dog_trh:string[1];
        medeni:medenihal;
        procedure baslat;
        procedure yenile(durum:medenihal);
        procedure goster;
end;
```

"Kisi" nesnesi, kişi ile ilgili bilgileri barındıran ve içindeki bilgilerle işlem yapan, iki prosedüre sahiptir. Nesnenin, kullanılabilmesi için, değişken olarak tanımlanması da gerekir.

```
Var
AKisi:Kisi;  Begin  Akisi.baslat;  end;
```

Akisi.baslat Kişi nesnesi içindeki baslat prosedürünü çalıştırmamızı sağlar. Kayıt alanlarına da aynı şekilde erişilebilir.

```
Akisi.isim:='Ahmet';
Akisi.soyad:='Arslan';
Akisi.dog_trh:='04/10/1960';
```

Bir nesne, diğer bir nesneden çok az farklı ise veya biri diğerinden daha genişse, bu iki nesneyi ayrı ayrı yazmaya gerek yoktur. Daha geniş olan nesne, diğer nesnenin türetilmişi olarak gösterilebilir ve sadece farklı kısımları eklenir. Bu durum kalıtım (Inheritance) özelliğinin sonucu olup, hen kodlamayı kısaltır, hem de işlemlerde kolaylık sağlar. Bir kişinin, aynı zamanda memur olduğunu düşünelim. Memur da bir kişi olduğundan "memur" nesnesi özelliklerini "Kisi" nesnesinden miras almış demektir.

```
Type
memur=Object(Kisi)
derece:string;     yıl:byte;
procedure baslat;
procedure yıl_yenile(yeniyıl:byte);
procedur derece_yenile(yeniderece:String);
end;
```

Yukarıdaki kodlamada görünmemesine rağmen, "memur" nesnesi aynı zamanda kişisel bilgilerle ilgili prosedürleri de içermektedir.

Nesneler içinde, iki türlü, değişken tanımlaması yapılabilir. Genel (public) alanda, değişkenler açıkça yazılır. Özel (private) alanda ise, sadece nesne içindeki diğer prosedürlerin erişebileceği değişkenler vardır. Aşağıda tanımlayacağımız "yas" nesnesine, yalnızca o nesne içindeki diğer prosedürler erişebilir.

```
Type
memur=Object(Kisi)
        derece:string; yıl:byte;
        procedure baslat;
        procedure yıl_yenile(yeniyıl:byte);
        procedur derece_yenile(yeniderece:String);
        private
        yas:byte;
end;
```

Bir unit, iki ana bölümden oluşmaktadır. Bunlar "interface" ve "implemantation" olarak adlandırılan bölümlerdir. Nesnelerin, başka programlarda kullanılması da mümkün olduğundan, bunun yapılması istendiğinde, nesnelerin unit içerisinde "interface" kısmına yazılmaları gerekir. Nesneye ait kodlar ise, "implemantation" kısmına yazılmalıdır. Aşağıda, daha önce verdiğimiz nesnelerin birer unit içinde kullanım şekilleri verilmektedir.

```
{ KISILER.PAS }
Unit kisiler;
INTERFACE
Const  Evli=True;      Bekar=False;
Type
        Medenihal=Boolean;        Kisi=Object
        isim:string[30];  soyad:string[20];  yas:integer;
        medeni:medenihal;
        procedure baslat;
        procedure yenile(durum:medenihal);
        procedure yaslandır;
        procedure goster;
end;
```

```
IMPLEMANTATION
Procedure Kisi.baslat;
begin
        isim:='';       soyad:='';      yas:=0;         medeni:=bekar;
end;
Procedure Kisi.yenile(durum:medeneihal);
begin
        medeni:=durum;
end;
Procedure Kisi.yaslandır;
begin
        yas:=yas+1; writeln('Aşağıdaki kayıt yenilenmiştir.');   goster;
end;
Procedure Kisi.goster;
begin
        writeln('İsim:'+isim+'Soyad:'+soyad);      writeln('Yaşı:',yas);
writeln('Medeni Hali:');
        if medeni=Evli then writeln('Evli') else writeln('Bekar');
end;
end.

{ MEMURLAR.PAS }
Unit memurlar;
INTERFACE
Type
        memur=Object(Kisi)
        derece:string;      yıl:byte;
        procedure baslat;
        procedure yıl_gir(yeniyıl:byte);
        procedur derece_gir(yeniderece:String);
        procedure goster;
end;

IMPLEMANTATION
Procedure memur.baslat;
begin
        derece:='';     yil:='';
end;
```

```
Procedure memur.yıl_gir(yeniyıl:byte);
begin
        yıl:=yeniyıl;
end;
Procedure memur.derece_gir(yeniderece:string);
begin
        derece:=yeniderece;
end;
Procedure memur.goster;
begin
        writeln('İsim:'+isim+'Soyad:'+soyad);     writeln('Derece:'+derece);
        writeln('Çalışma Yılı:',yıl);
        writeln;
end;
end.
```

İki unitin, bir ana programda kullanılma esasları da şu şekildedir:

```
{ DENEME.PAS }
Program Deneme;
Uses KISILER,MEMURLAR;
Var     memur1:memur;
begin
        memur1.baslat;
        with memur1 do
        begin
                isim:='can';   soyad:='canan';     medeni:=bekar;
                yas:=33;       derece_gir('3/3');  yil_gir(10);
        end;
        memur1.yaslandır;
        readln;
end.
```

Bu program "Memur" nesnesinin atasında bulunan yaşlandır prosedürünü çağırarak bitiyor. Bu programın çıktısı şu şekildedir:

```
Aşağıdaki Kayıt yenilenmiştir.
İsim:Can      Soyad:Canan              Yaşı:34      Medeni Hali:Bekar
```

Bu "Memur" nesnesinin çıktısı değildir. "Kisi" nesnesinin çıktısıdır. "Memur" nesnesinin çıktısının alınabilmesi için "constructor" yani yapıcı, kullanılmalıdır ve aynı isimdeki prosedürlerin yanına "virtual" yazılarak, sanal oldukları belirtilmelidir. Bu durumda KISILER.PAS ve MEMURLAR.PAS unitlerinin interface kısımları şu şekilde değişecektir.

```
Kisi=Object
        isim:string[30];  soyad:string[20];  yas:integer;
        medeni:medenihal;
        constructor baslat;
        procedure yenile(durum:medenihal);
        procedure yaslandır;
        procedure goster;virtual;
end;

Memur=Object(Kisi)
        derece:string;     yıl:byte;
        constructor baslat;
        procedure yıl_gir(yeniyıl:byte);
        procedur derece_gir(yeniderece:String);
        procedure goster;virtual;
end;
```

Programın yeni çıktısı ise şu şekilde olur:

```
Aşağıdaki kayıt yenilenmiştir.
İsim:Can      Soyad:Canan      Derece:3/3
Çalışma Yılı:10
```

Bu sonuç, "Memur" nesnesine ait olan göster prosedürünün sonucudur.

Tablo 2.10. Örnek Bir Turbo Pascal Programı

Sonuç olarak, ata nesneye ("Kisi" nesnesi) ait bir prosedür, türetildiği nesne içerisinde, ata nesneden türetilen nesneye ait yordamları çağırabilmiştir. Bu nesneye yönelik programlamanın temelinde bulunan ve bütün uygulamalarada en çok kullanılan bir özelliktir. Torun nesne ata nesneyi kapsamaktadır.[102]

---

[102] Smedema, C. H. s. 156

Üçüncü Bölüm

KULLANILAN DONANIM - YAZILIM VE GELİŞTİRİLEN NESNEYE YÖNELİK ÇİZİM EDİTÖRÜ

Bu bölümde, bilgisayarların grafik özellikleri ve Turbo Pascal nesneye yönelik programlama dilinin, grafik alanında kullanımına ait, bilgiler verilmektedir. Donanım ve yazılım birimleri ile ilgili bu bilgilerden sonra, nesneye yönelik olarak tasarlanan, çizim editörünün nasıl geliştirildiği, hangi unitlerin ve dosyaların kullanıldığı, bunların birbirleri ile ilişkileri ve akış şemaları verilmekte, en son olarak da, geliştirilen örnek çizim projesinin kullanım şekli anlatılmaktadır.

3.1. KULLANILAN DONANIM - YAZILIM

Bilgisayarın iki temel bileşeni, donanım ve yazılımdır. Teknolojide meydana gelen değişim farklılıkları, bu iki bileşen arasında farklı teknolojik seviyelerin oluşmasına neden olsa da, genelde birbirlerini yakından takip edebimektedirler. Yazılım alanında son yıllarda oluşan gelişme, nesneye yönelik yaklaşım tekniği sayesinde olmuştur. Bu tekniğe uygun ve üst düzey programlama dillerinden biri olarak geliştirilen Turbo Pascal nesneye yönelik programlama dili, PC'lerdeki (personal computer), görüntüleme imkanları ve grafik kullanım özelliklerinden yararlanılarak, grafiksel işlemlerin gerçekleştirilmesinde, uygun ortamın oluşturulmasına imkan vermektedir.

3.1.1. Görüntüleme Donanım Sistemleri ve PC Grafik Kullanımı

Bilgisayarda grafiğin ilk örnekleri, modern bilgisayar çağının ilk günlerinde gerçekleştirildi. 1950'lerin başında, Massachuset Teknoloji Enistitüsü'nde bir grup araştırmacı tarafından katot-ışınlı tüp içinde bir elektron ışınının sapmasını kontrol etmek için çalışmalar yapıldı.[103] 1960'lara gelindiğinde oldukça önemli gelişmeler gerçekleşmişti. Bu yıllarda, doğrudan görüntü korumalı katot tüpü olarak adlandırılan standart bir görüntü cihazı oluşmuştur. Bilgisayarda grafiğin bir sonraki çağı, 1970'lerde rastel grafiğin

---

[103] Margaret, M, B. ve Allen, L, A. "Declarative Visual Languages". **Journal Of Visual Languages and Computing.** No:5. USA, Academic Press Limited, 1994, s. 143

gelişi ile karakterize edilir. Satır taramalı grafik görüntülemesi geliştirilmiş ve vektörler yerine pixel dizileri oluşturulmuştur. Bu pixel'ler özel bir bellek alanında saklanırlar ve görüntüleme donanımı tarafından saniyede 50 veya 70 kez taranarak, görüntüyü satır satır yenilerler. 1980'lerden sonra güçlü iş istasyonları aracılığı ile mühendislerin, bilim adamlarının, garfikerlerin ve mimarların standart yardımcı araçları grafik kullanımı olmuştur. CAD (Computer Aided Design) sistemi kullanıcıları, günümüzde, etkileşimli, üç boyutlu, katı modelleyicilerle çalışmaktadırlar. Animatörler ve TV-reklam ressamları, mevcut donanım ve yazılımları kullanarak gerçeğe yakın resimleri oluşturabilmektedirler.[104]

3.1.1.1. Görüntüleme Donanımı

Görüntüleme adaptörü, görüntü belleğindeki alfanümerik ve grafiksel bilgilerin, monitörde görüntülenmesini sağlayan bir sistemdir. Görüntüleme donanımı ile etkileşimli programların yazımında, görüntüleme sisteminin programlanabilir elemanları arasındaki ilişkinin bilinmesi çok gerekli değildir. Bunlar sadece programcıya yardımcı olan bilgilerdir. Ayrıca iyi bir grafik programı yazmak için, donanımın devre dizaynının bilinmesine de ihtiyaç yoktur. Görüntüleme sisteminin belli başlı elemanları şunlardır:

1. Görüntü Belleği: Görüntüleme sistemi, görüntü belleğindeki tüm veriyi okuyarak ekrana görüntüler. Bu işlemi sürekli olarak ve tekrar tekrar yaparak ekrandaki görüntüyü yeniler. Görüntü belleğindeki her bit veya bit grubu, ekrandaki belirli bir noktanın (pixel) rengini ve parlaklığını belirtir. Bir görüntü belleğinin mevcut RAM'daki (Random Access Memory) gerçek boyutu, görüntüleme sisteminin cinsine göre değişiklik gösterir.

2. Monitör: Bir bilgisayarın görüntüleme donanımının en görünür parçası monitördür. Bununla beraber monitör, donanımda doğrudan pragramla kontrol edilebilen bir cihaz değildir. Daha çok programlanabilir donanımı içeren bir görüntüleme alt sistemidir. Bugün kullanılan tüm monitörler satır taramalı cihazlardır. Bir elektron ışını, ekranın sol üst köşesinden başlayarak soldan

---

104 Li,S. ve Shahidehpour,S,M. "An Object Oriented Power System Graphics Package for Personal Computer Environment". **IEEE Transactions on Power Systems.** No:3. USA, IEEE, 1992, s. 1054

sağa her satırı sırayla tarar. Işın her satırı tararken ekrandaki noktaların renk ve parlaklığı değişir ve bütün satır sabit bir şekil alarak görüntülenmiş olur.

3. CRT Denetleyici: CRT denetleyici (Cathed Ray Tube Controller - CRTC), ekran için yatay ve düşey tarama zamanlama sinyalleri meydana getirir. Aynı zamanda zamanlama sinyalleri ile, eş zamanlı olarak görüntü belleği adres sayacını, 1 arttırır. Görüntüleme sistemi, CRTC'nin adres değerlerini kullanarak görüntü belleğinden bilgiler okur, kodlarını çözer ve monitöre sinyaller şeklinde gönderir. Bu yolla CRTC, görüntülemeyi idare eden zamanlama sinyalleri ile görüntü belleğindeki bilgilerin görüntülenmesini yerine getirir.

4. Renk Görüntüleme Donanımı: Bir görüntüleme modunda, görüntülenebilir renklerin sayısı ve çeşidi görüntüleme sisteminin renk-kod çözücüsünün dizaynına ve görüntü sinyal üretici elemanlarına bağlıdır. Renk kod çözücüsü, üretilen renk ve parlaklık bilgilerini görüntü sinyal üreticisi tarafından kontrol edilen sinyallere çevirebilmesi için görüntü belleğinde depolanan verileri kullanır.

5. Mod Kontrolü: Görüntü belleğindeki bilgilerin ne şekilde tutulacağı, aynı zamanda bir görüntüleme modunu karakterize eder. Tüm PC ve PS/2 sistemlerinde, görüntüleme modları görüntü belleğinin veri formatına göre grafik veya alfanümerik olarak sınıflandırılabilir. Her görüntüleme modu, onun çözünürlüğü (resolution) ile görüntülenebilen pixel'lerin veya karakterlerin sayısı ile karakterize edilebilir. Bir görüntüleme modunda yatay ve düşey çözünürlük, monitörün yatay ve düşey tarama oranına ilaveten, monitör ekranındaki nokta oranının bir fonksiyonudur. Görüntülenen pixel'lerin sayısı, nokta oranı ve düşey tarama oranına benzerdir. Gerçek yatay ve düşey çözünürlük, yatay tarama oranına bağlıdır.

3.1.1.2. Grafik Adaptörleri

Grafik adaptörleri, görüntü belleğinde bit grupları şeklinde yer alan grafiksel verilerin her birini ekranda bir noktaya (pixel) karşılık gelecek şekilde görüntülenmesini sağlayan bir sistemdir. Grafik adaptörleri genelde tek renkli monitör ve çok renkli monitör için olmak üzere iki sınıfa ayrılırlar. Fakat bunlar da kendi aralarında destekledikleri ekran çözünürlüğü ve renk sayısı

bakımından farklılıklar gösterirler. Günümüze kadar ve halen en yaygın kullanılan grafik adaptörleri şunlardır: [105]

1. Monochrom Display Adapter (MDA)
2. Color Graphics Adapter (CGA)
3. Enhanced Graphics Adapter (EGA)
4. Video Graphics Adapter (VGA)
5. Hercules Graphics Card (HGC)
6. Hercules InColor Card (HCC)

Bir PC de, grafik çizimlerinin gerçekleştirilebilmesi için ilk şart, donanımda grafik sistemini oluşturan grafik adaptörünün var olmasıdır. Yukarıda saydığımız yaygın olarak kullanılan grafik adaptörlerinden, herhangi birinin bulunmasından sonra grafik kullanımına geçilebilir. Kullanılan bu adaptörler genel olarak üç farklı özelliğe sahiptirler.[106]

1. Çizimin yapıldığı, grafik ekranın çözünürlüğü
2. Çözümde kullanılabilen renklerin sayısı
3. Gösterilebilen bir ekranlık bellek sayfalarından, kaç tane olduğu

Grafik adaptörlerinin yüksek pixel alanına sahip olmaları çizim kalitesinin artmasını sağlarken, çizimde kullanılabilen renklerin fazlalığı grafiğin görüntü kalitesini arttırır. Grafik çizim alanının birden fazla sayfadan oluşması ise, grafik animasyonlarında kolaylık sağlar.

Tek bir grafik adaptörünü destekleyen program yazmak yerine, grafik paketi tarafından desteklenen tüm grafik adaptör ve modlarında çalışabilecek bir program yazmak için, grafik adaptör ve modunun program tarafından otomatik olarak belirlenmesi gerekir. Turbo Pascal editöründe bu işlem detectgraph komutu ile gerçekleştirilir.

```
detectgraph ( GrAdapter, GrMod); veya
initgraph ( GrAdapter, GrMod, ' ');
```

---

[105] Erdun,Hakan. **Turbo ve Borland C & Pascal ile Grafik. İstanbul,** Beta Basım Yayım, 1993, ss. 3-17

[106] Yanık, Memik. **Borland Delphi ile Görsel Programcılık.** İstanbul, Sistem Yayıncılık, 1996, s. 117

Bilgisayarın grafik adaptörü ve modunun belirlenmesinden sonra yapılacak ilk iş, çizim için grafik sisteminin başlatılmasıdır. Grafik sisteminin başlatılması grafik adaptörünün çizim için hazır duruma getirilmesi demektir. Bunun için Graph unit'inin aktif duruma getirilmesi gerekmektedir. Bu sayede verilen grafik adaptörü ve moduna ait bilgileri içeren uygun grafik arabirim dosyası (BGI uzantılı dosyalar) hafızaya yüklenerek, grafik arabirimi arasındaki iletişimi hazırlar ve grafik sistemini başlatır.

3.1.2. Turbo Pascal Programlama Dili ve Grafik Özellikleri

Turbo Pascal programlama dili, nesneye yönelik yaklaşım kurallarına uygun, modüler bir dildir. Turbo Pascal ile üretilen kodlamaların, benzer projelerde yeniden kullanılmaları ve yine üretilen kodların, derlendikten sonra, bir başka bilgisayarda çalıştırılabilmeleri (taşınabilmeleri) mümkündür.

Turbo Pascal programlama dili, aynı zamanda, grafiksel işlemlerin kullanılmasında da oldukça önemli özelliklere sahiptir. Bu amaçla geliştirilmiş bir çok unit, prosedür, fonksiyon ve alt program kullanımına müsaade eder. Birbirinden farklı, 16 adet renk kodu vardır.

Geliştirilen örnek çizim editörü projesinin tanıtılmasına geçmeden önce, Turbo Pascal nesneye yönelik programlama dilinin temel özellikleri aşağıdaki kesimlerde ele alınmaktadır.

3.1.2.1. Temel Yapı

Turbo Pascal nesneye Yönelik Programlama dilinde mevcut özellikler ve uyulması gereken kurallar, programlama dilinin yapısı esas alınarak şu şekilde tasnif edilebilir:

Bir Turbo Pascal programı, en temel anlamda üç ayrı kısımdan oluşur. Bunlar sırası ile aşağıdaki gibidir:

1. Program Başlığı (Heading): PROGRAM ifadesi ile verilen, en fazla 30 karakter olabilen programın amacını açıklayıcı tanım adı. Tanıtım amaçlı olduğundan kullanılması zorunlu değildir.

2. Bilgi Tanıtım Bölümü (Declarations): Programın işlem bölümünde bilgi işleme tabi tutulacak tüm bilgiler bu bölümde tanımlanmalıdır. Tanımlamalar belirli bloklar halinde yapılır. Tanımlama bloklarının herbiri, özel bir bildiri deyimi ile başlar. Bildiri deyimleri kullanılarak oluşturabilecek tanımlama blokları Tablo 3.1'de gösterilmektedir.

UNIT	: Harici programların tanıtım bloğu
USES	: Kullanılacak arşiv programların tanıtım bloğu
LABEL	: Etiket tanımlama bloğu
CONST	: Sabit isimleri tanımlama bloğu
TYPE	: Bilgi tanımlama bloğu
VAR	: Değişken tanımlama bloğu
PROCEDURE	: Alt program tanımlama bloğu
FUNCTION	: Fonksiyonel alt program tanımlama bloğu

Tablo 3.1. Turbo Pascal'da Kullanılan Tanımlama Blokları

3. Program İşlem Blokları: Programın amacına yönelik yapılacak işlemlerin bloklar halinde gerçekleştirildiği bölümdür. Bir programda en az bir işlem bloğu bulunur. Ana program işlem bloğunun sonunda bulunan ifade END. şeklindedir. Diğer blokların sonunda kullanılan END ifadesi ise ';' ile bitirilir. En basit haliyle bir turbo pascal programı işlem bloğu şu şekildedir.[107]

```
Begin
        işlem bloğu
End.
```

3.1.2.2. Veri Tipleri

Veri tipleri, program içerisinde kullanılacak sayısal veya alfasayısal bilgi tiplerinin tanımlarında kullanılan ifadelerdir. TYPE, VAR, PROCEDURE ve FUNCTION bloklarında tanımlanan bilgi tip tanımları üç genel gruba ayrılırlar:

1. Sayısal Veri Tipleri: Tam sayı ve ondalıklı say bilgileri şeklinde kendi içinde tasnif edilebilen, sayısal bilgilerin tanımlandıkları veri tipleridir. Her veri tipi için verilen limitler arasında değer aktarılması yapılmalıdır. Aksi halde

---

[107] Altan,Naci. **Turbo Pascal 7.0.** İstanbul, Alfa Basım Yayım, 1995, s. 86

taşma hatası (overflow error) oluşur ve programın çalışması durur. Kullanlabilecek sayısal veri tipleri şunlardır.

Byte : 0-255 arasında işaretsiz 256 değeri, bellekte 1 byte uzunluğunda bir alanda saklayan bilgi tipi.

Shortint : -128 ile +127 arasındaki işaretli değerleri, bellekte 1 byte uzunluğunda bir alanda saklayan bilgi tipidir.

Integer : -32768 ile 32767 arasındaki işaretli sayısal değerleri, 2 byte uzunluğunda bir alanda saklayan bilgi tipidir.

Word : 0 ile 65535 arasındaki işaretsiz sayısal değerleri, 2 byte uzunluğunda bir alanda saklayan bilgi tipidir.

Longint : -2147438648 ile 2147438647 arasında işaretli sayısal değerleri, bellekte 4 byte uzunluğunda bir alanda saklayan bilgi tipidir.

Aritmetik işlemlerde, tüm tam sayı veri tiplerinde bilgi aktarımı yapılabilir. Örneğin, byte değişkenindeki bilgi, byte limitini aşmadığı sürece Longint değişkenine aktarılabilir.

Real : Ondalıklı sayılar için kullanılan bu değişken veri tipi, 11 veya 12 haneli rakamlarda duyarlıdır. Bu, değişkenlere 2.9*10E-39 ile 1.7*10E+38 arasında aktarılabilecek işaretli bir değeri, bellekte 6 byte uzunluğunda saklayan veri tipidir.

Single : Short Real olarak ta adlandırılan bu değişken veri tipi, 7 veya 8 haneli rakamlarda duyarlıdır. Aktarılan sayısal değerleri bellekte 4 byte uzunluğunda bir alanda saklar.

Double : Long Real olarak ta adlandırılan bu değişken veri tipi, 15 veya 16 haneli rakamlarda duyarlıdır. Aktarılan bilgileri, bellekte 8 byte uzunluğunda bir alanda saklar.

Extended : 19 veya 20 haneli rakamlarda duyarlı olan bu ondalıklı değişken veri tipi, aktarılan bilgileri 10 byte uzunluğunda bir bellek alanında saklar.

2. Alfasayısal Veri Tipleri: Alfasayısal bilgilerin tanımlanmasında kullanılan bilgi tipleridir.

Boolean : Bir mantıksal işlem sonucunun kontrolünde kullanılan, true (doğru) ve false (yanlış) değerleri alan bilgi tipidir. Genellikle şart cümlelerinde kullanılır.

Char : Bellekte 1 byte uzunlukta bir alan kaplayan, ASCII kodlu bilgilerin tanımlanmasında kullanılan, veri tipidir. Tek bir karakteri tanımlarlar. Bu değişken veri tiplerine 0 ile 255 arası ASCII bir değer aktarılabilir.

Strings : Program içinde kullanılacak alfasayısal bilgilerin, aktarılacakları değişken veya saha isim tiplerinin uzunluklarıyla birlikte tanımlanır. String bilgilerin aktarılacakları adreslere, 1 ile 255 arasında bir uzunluk verilebilir. String değişkenin 0. elemanı, pascal derleyicisi tarafından değişken uzunluğu için kullanılır ve length byte olarak adlandırılır. Veri tipi tanımında kaç uzunluk verilmişse, aktarılacak bilgi bu uzunluğu geçmemelidir.

3. Birlikte Kullanılabilen Farklı Tipler: Program yapısı içerisinde birbirlerinden farklı tiplerde tanımlanan bilgi tipleri arasındaki, kontrol kullanım şekline "Type Casting" denir. Bu ortamda hem sayısal hem de alfasayısal veri tiplerinin kullanımı mümkün olmaktadır.

3.1.2.3. Turbo Pascal Standart Unitleri

Turbo Pascal nesneye yönelik programlama dili, en önemli özelliğini unit'lerden almaktadır. Unitler, Turbo Pascal programlarında gerek hazır olarak verilen gerekse programcılar tarafından oluşturulan, hazır paket programlardır. Her bir unit kendi içerisinde prosedür, fonksiyon, değişken ve sabit bilgilerden oluşur.[108] Herhangi bir unit, ana programdan USES komutu kullanılarak aktif duruma getirilir. Turbo Pascal standart unitlerinin yanısıra programcılar tarafından oluşturulacak unitler de standart unitler gibi işlem görürler. Ayrıca bir unit içersinden de diğer bir unit aktif duruma getirilebilir. Programcı, böyle bir paket oluştururken diğer programların veya başka unitlerin erişebilmesi için gerekli tanımlamaları INTERFACE kesiminde; yerel işlem prosedürlerinin ve işlem değişkenlerinin tanımlarını IMPLEMENTATION kesiminde yapmalıdır. Yazılan her unit derleyiciden geçirilince, .TPU dosyaları pascal tarafından oluşturulur. Programcılar tarafından geliştirilen hazır programlar, programcıyı ayrıca ilgili prosedürleri yazmaktan kurtarır ve bellekte yer kazandırır.

Turbo Pascal standart unitleri Tablo 3.2'de verilmektedir.

CRT	: Ekranla ilgili kullanılan komutlar uniti
DOS	: İşletim sistemiyle kullanılacak komutlar uniti
GRAPH	: Grafik çizimleri için hazırlanan unit
GRAPH3	: Turbo 3. versiyonları için hazırlanan grafik desteği uniti
OVERLAY	: Overlay uniti

[108] Aho, A. V. Sethi, R. ve Ullman, J. D. **Compilers: Principles, Techniques, and Tools.** New York, Addison-Wesley, 1986, s.203

PRINTER : Yazıcıyla ilgiili komutlar uniti
SYSTEM : Bilgisayar bellek yapısı ile ilgili unit
TURBO3 : Turbo Pascal 3. versiyon farklılıkları uniti
WINDOS : Windows altında yapılacak çalışmalar için hazırlanmış unit

Tablo 3.2. Turbo Pascal Standart Unitleri

3.1.2.4. Dos ve Windows Çalışmalarında Grafik

Turbo Pascal, zengin grafik prosedür ve fonksiyonlarıyla grafik programlarını daha kolay yapılabilir duruma getirmiştir. İki çeşit grafik desteği sağlamaktadır:

1. BGI (Borland Graphics Interface - Borland grafik arabirim kartı) DOS destekli çalışmalarda tüm grafik işlemlerini destekler.

2. GDI (Graphic Device Interface - Grafik sürücü arabirim kartı) WINDOWS uygulamaları için kullanılır.

Normalde kişisel bilgisayarlarda, yazı modu olarak adlandırılan Text mod çalışmalarında, ekranın 25x80 matrislik alanında, ASCII karakterler, rahatlıkla görüntülendirilebilirler. Halbuki, grafik modunda çok farklı tanımlamalar gerekmektedir. Bir karakter yerine, bilgisayar ekranına, pixel denilen küçük noktalardan oluşan şekiller girilir ve ekrandaki tek bir karakter, bir çok pixelden oluşur.

Ekranlar, pixellerden oluşan yatay ve dikey satırlara ayrılmıştır. Ancak bilgisayar ekranlarında pixel sayıları farklı olabilir, CGA alçak çözünürlük modu, 320 yatay, 200 dikey pixel'e sahiptir. VGA yüksek çözünürlük modunda, 640 yatay ve 480 dikey pixel mevcuttur. Pixel sayısının çoğalması, görüntü kalitesini arttırır. Ekranın sol üst köşesi 0,0 ve sağ alt köşesi ise 319,199 veya 639, 479 koordinatlarındadır.

3.1.2.5. Renk ve Model Kullanımları

Model çizimlerinde ve çizimlerin renklendirilmelerinde, Graph unit kullanıcıları için, yirmi'nin üzerinde alt program hazırlanmıştır. Her biri, çizdirilen şeklin yapısına göre, iç kısımlarını renklendirir.

FillEllipse : Verilen x ve y koordinat merkezli elips çizer.
FillPoly : Belirtilen noktalarda çizilen poligonun içini doldurur.
FloodFill : Kenarları kapalı geometrik bir şeklin içini boyar.
GetBkColor : Zemin rengini belirler.
GetColor : Grafik modun çizilen rengini belirler.
GetDefaultPalette : Grafik sürücünün renk palet tipini tanımlar.
GetFillSettings : Geometrik şeklin biçimini ve rengini belirtir.
GetLineSettings : Mevcut satırın şeklini, biçimini ve kalınlığını belirler.
GetMaxColor : Aktif durumdaki palet için en yüksek renk değerini belirler.
GetPalette : Aktif palet değerini tesbit eder.
GetPaletteSize : Grafik modun destekleyebildiği maksimum palet numarası.
SetAllPalette : Belirtilen tanımlara göre tüm palet renklerini değiştirir.
SetBkColor : Verilen renk koduna göre zemin rengini değiştirir.
SetColor : Belirtilen renk koduna göre çizim rengini düzenler.
SetFillStyle : Belirtilen biçim ve renkte geometrik şeklin içinin tarar.
SetLine Stiyle : Geometrik şeklin satır stilini ve genişliğini düzenler.

3.1.2.6. Çizgiler ve Geometrik Şekiller Çizdirme

En basit haliyle bir grafik çizimi, grafik ekrandaki pixellerin istenilen renkte boyanmasıyla yapılır. Bir pixel'in istenilen renkte boyanması için, üç bilgiye ihtiyaç vardır:

1. Piksel'in kaçıncı sütuna karşılık geldiği - (x)
2. Piksel'in kaçıncı satıra karşılık geldiği - (y)
3. Pikselin hangi renge boyanacağı - (renk)

Pascal'da GRAPH.TPU unit'inde tanımlı bulunan sabit isimleri veya bunlara karşılık gelen tamsayı renk kodları kullanılır. Kullanılabilecek renkler ve kodları Tablo 3.3'de gösterilmektedir.

Sabit ismi	Değeri	Anlamı
BLACK	0	Siyah
BLUE	1	Mavi
GREEN	2	Yeşil
CYAN	3	Yeşilli mavi
RED	4	Kırmızı
MAGENTA	5	Morumsu kırmızı
BROWN	6	Kahverengi
LIGHTGRAY	7	Açık gri
DARKGRAY	8	Gri
LIGHTBLUE	9	Açık mavi
LIGHTGREEN	10	Açık yeşil
LIGHTCYAN	11	Açık yeşilimsi mavi
LIGHTRED	12	Açık kırmızı
LIGHTMAGENTA	13	Açık morumsu kırmızı
YELLOW	14	Sarı
WHITE	15	Beyaz

Tablo 3.3. Turbo Pascal'da Kullanılabilecek Renkler ve Kodları

Temel çizim komutları ise şunlardır:

Doğru parçası, dikdörtgen, daire, elips vb. şekiller grafik çizimlerinde oldukça sık kullanılırlar ve karmaşık grafik çizimlerinin temel elemanlarını teşkil ederler.

Line komutu : Bir doğru parçasını çizmek için kullanılır. Doğru parçasının çiziminde başlangıç (x1,y1) ve bitiş (x2,y2) koordinatları verilmelidir.

Circle komutu: Daire çizimi amacıyla kullanılır. Çizim için önce dairenin merkezi koordinatları (x,y) sonrada yarıçapı (r) verilmelidir.

Arc komutu : Bir daire parçası çizmek için kullanılır. Çizim için önce daire parçasının merkezi koordinatları (x,y), sonra yayın başlangıç ve bitiş açıları (acı1,acı2) ve son olarakta (r) daire yayının yarıçapı verilir.

Elipse komutu: Bir elips veya eliptik yay çizimi için kullanılır. Elipste verilen parametrelerin anlamları arc'taki ile benzerdir. Sırasıyla (x,y) elips merkezinin koordinatları, (acı1,acı2) elips yayının başlangıç ve bitiş açıları, (rx,ry) elipsin x ve y yönlerindeki yarıçaplarıdır.

Rectangle komutu: Dikdörtgen çizimi amacıyla kullanılır. Parametrelerden (x1,y1) dikdörtgenin sol-üst köşesinin koordinatlarını, (x2,y2) de sağ-alt köşesinin koordinatlarını belirler.

3.1.2.7. Yazı Fontlarında Grafik Desteği

Normalde grafik modu, yazı fontlarıyla pek kullanılmaz. Grafik ekranı ile metin ekranı birbirlerinden farklıdırlar. Ancak grafik ve yazı fontlarının kombine kullanımı, masa üstü çalışmaları için çok önemlidir. Turbo pascal'da write ve benzeri komutlarla metin ekranında yazı yazmak EGA ve VGA hariç, diğer grafik sistemleri için mümkün değildir. Grafik sisteminin bu özelliğinden dolayı grafik ekranda yazı yazımı için özel teknikler kullanılır.[109]

Bu teknikler iki tanedir:

1. Nokta temelli karakter tanımlama tekniği : Nokta temelli karakter tanımlamasında, tanımlama amacıyla kullanılan pixel alanının boyutu, her piksel bir bit'e karşılık gelecek şekilde 8x8 şeklindedir. Nokta temelli tanımlama tekniği ile içi belirli bir düzenle doldurulmuş dikdörtgen pixel alanının ekrana basılmasıyla ilgili karakterin görüntülenmesi ile sağlanır.

Örneğin 'A' karakterini grafik ekranda basmak için 8x8 bitlik bir alanın içi A karakterine uygun 0 ve 1 ile doldurulur. Bit'in 0 olması pikselin boyanmıyacağı, 1 olması pikselin boyanacağını belirtir.

2. Vektör temelli karakter tanımlama tekniği : Vektör temelli teknikle karakter tanımlaması, basitçe çizim komutları kullanılarak karakterin ekranda çizilmesidir. Bir grafik nasıl oluşturuluyorsa aynı şekilde bir grafik parçası gibi oluşturulur. Vektör tanımlama tekniği ile bir karakteri oluşturmak için, birinci teknikteki dikdörtgen alan piksellerinin doldurulması yerine, harhangi bir boyuta sahip dikdörtgen alanında karakterin çiziminde gerekli piksel koordinatları kullanılır.

---

[109] Erdun, Hakan. s.34

Örneğin ekrana 'A' karakterini basmak için, uygun büyüklükte bir alan seçilerek 'A' karakteri ile ilgili uygun piksel koordinatları belirlenir. Bu işlem her bir karakter için yapılarak sadece koordinat değerlerinin bulunduğu bir veri seti oluşturulur. Metnin ekrana basılması sırasında, karakterin çizim koordinatlarını içeren bu veri setinden ilgili karakterlere ait koordinat değerleri alınır. Daha sonra bu koordinatların araları doğrularla birleştirilerek ekrana metnin basımı sağlanır.

Vektör temelli tanımlama tekniği, nokta temelli tanımlama tekniğinden çok üstündür. Bu fark, karekterin boyutu büyütüldüğünde açıkça ortaya çıkmaktadır. Vektör temelli teknikle tanımlanan karakter alanı ne kadar büyük olursa olsun karakter hep aynı kalitede çizilecektir. Nokta temelli teknikte ise, piksel alanı büyütüldüğünde içi boyalı pikseller de o derece büyültülecek, sonuçta ekranda kaba bir görünüm ortaya çıkacaktır.

Bu teknikle çeşitli karakter setlerinin oluşturulması, oldukça zahmetli ve sıkıcı bir işlemdir. Turbo pascal grafik paketi, bu yükü programcının üzerinden kaldırmak için önceden, vektör temelli karakter tanımlama tekniği ile hazırlanmış 11 çeşit karakter setini programcılara hazır olarak sunmuştur. Bu karakter font setleri, oluşturulan yazı tipine uygun olarak verilen bir isimle diskte ".CHR" uzantısıyla depolanmışlardır. Bu dosyalara kısaca karakter font dosyaları denir. Bu dosyalar şunlardır:

GOTH.CHR	:	Gotik yazı font seti
LITT.CHR	:	Küçük yazı font seti
SANS.CHR	:	Sans Serif font seti
TRIP.CHR	:	Tripleks font seti
BOLD.CHR	:	Kalın çizgili yazı font seti
EURO.CHR	:	Avrupa yazı font seti
LCOM.CHR	:	Kompleks yazı font seti
SCRI.CHR	:	El yazısı font seti
SIMP.CHR	:	Basit yazı font seti

Bu dosyalara ilaveten, grafik sistemi içinde hazır olarak bulunan bir font seti daha vardır. Default olarak isimlendirilen bu font dosyası, nokta temelli karakter tanımlama tekniği ile oluşturulmuştur.

Metnin ekrana basımında kullanılacak olan font, karakter dosyasını belirten tamsayı bir değerdir. Bu değerler, bu değerleri temsil eden sabit isimler ve bunların hangi font dosyasını belirttiği, Tablo 3.4'de gösterilmiştir:

Turbo Pascal'da Sabit İsimleri	Değeri	Font Tipi Dosya İsmi
DEFAULT_FONT	0	8x8'lik font seti
TRIPLEX_FONT	1	TRIP.CHR
SMALL_FONT	2	LITT.CHR
SANS_SERIF_FONT	3	SANS.CHR
GOTHIC_FONT	4	GOTH.CHR
SCRIPT_FONT	5	SCRI.CHR
SIMPLEX_FONT	6	SIMP.CHR
TRIPLEX_SCR_FONT	7	TRIP.CHR
COMPLEX_FONT	8	LCOM.CHR
EUROPEAN_FONT	9	EURO.CHR
BOLD_FONT	10	BOLD.CHR

Tablo 3.4. Grafik Sistem Font Seti

3.1.2.8. Grafik Görüntü Saklama

Grafik düzenlemesi yapılan bir görüntü parçasını, kolaylıkla erişilebilecek bir yerde saklamak önemlidir. Çünkü elde edilen çizim parçasının yeniden elde edilebilmesi, hem mümkün olmayabilir, hem de çok emek ve zaman gerektirir. Elde edilen bir grafik parçasının saklanması üç aşamada gerçekleştirilebilir:[110]

1. Saklanacak görüntü için ne kadar bellek gerektiği hesaplanır. Bu tanımlamada ekranın sol üst köşesiyle sağ alt köşesi arasında verilen koordinatlardaki görüntü için gereken bellek, ImageSize fonksiyonu ile hesaplattırılır. ImageSize, konumu verilen görüntünün, bellekte kaplayacağı alanı (byte adetini), hesaplar. Saklanacak parça 64 KB'tan fazla olmamalıdır.

---

[110] Addyman, A. **Specification for the Computer Programming Language Pascal.** New York, ISO/DP, 1983, s. 81

2. Saklanacak görüntü alanı kadar dinamik bellekte yer ayrılır. Bunun için GetMem prosedürü, P pointer değişkeni ile bellekte hesaplanan alan kadar yer ayırır.

3. Görüntü, ayrılan geçici belleğe kaydedilir. Bunun için GetImage prosedürü kullanılır. Ayrılan belleğin P pointeri ile belirtilen konumuna, GetImage prosedürü görüntüyü kaydeder. Görüntü, dinamik belleğe pointer göstergesinin belirttiği konum adresine yazılır.

Görüntü, dinamik belleğe saklattırıldıktan sonra istenilen herhangi bir zaman PutImage prosedürü ile okutularak görüntülenir. PutImage prosedürü, dinamik bellekteki görüntüyü pointer göstergesi ile belirtilen konum adresinden okuyarak tekrar görüntülenmesini sağlar. Görüntünün şekli verilecek mantıksal operatörlerle belirtilir. PutImage ile kullanılan mantıksal operatörler Tablo 3.5'de gösterilmektedir.

Operatör	Değeri	Kullanıldığı Assembler Komutu
NormalPut	0	MOV
CopyPut	0	MOV
XORPut	1	XOR
OrPut	2	OR
AndPut	3	AND
NotPut	4	NOT

Tablo 3.5. Mantıksal Operatörler

3.1.2.9. Pointerler ve Nesne İçin Pointer Kullanımı

Pointer tipi veriler, iki ayrı tamsayıdan meydana gelen ve belli bir bellek lokasyonun adresini gösteren değerlerden oluşur. Pointer tipi verinin ilk değeri, gösterilen lokasyonun segment adresine, ikinci değeri ise offset adresine karşılık gelir. Bu tür veriler ile kullanılan bilgisayarın bütün belleğine direkt erişim sağlanabilir. Pointer tipi veriler, daha çok dinamik tipteki değişkenleri oluşturmak ve kullanmak için tercih edilirler. Amaç kullanılan bilgisayarın boş bellek alanlarından yararlanmaktır. Pointer tipi değişken kavramına geçmeden önce, değişkenleri statik ve dinamik olmak üzere iki ayrı sınıfta incelemek gerekir:

Statik değişkenler: Programcı tarafından tanımlanan bütün değişkenlerin statik değişken sınıfına girdiğini söylemek mümkündür. Bu tür değişkenlerin formları ve bellekte kaplayacakları alanın uzunluğu ayrıca böyle bir değişkenin bellekteki yeri program boyunca sabittir. Yani programcı VAR bölümünde bir değişken tanımlamış ise, bu değişkenin tipine ve uzunluğuna program boyunca uymak zorundadır. Programın herhangi bir noktasında, değişkenin tipi, bellekte kapladığı alan ve bellekteki yeri değiştirilemez. Ayrıca statik değişkenler için Turbo Pascal tarafından ayrılan alan, bu değişkenin kullanımı sona erdikten sonra yok edilemez. Değişken için bellekte ayrılan alan, program sona erinceye kadar korunur.

Dinamik değişkenler: İcra sırasında formu ve uzunluğu değiştirilebilen değişkenlere ihtiyaç duyulur. Bu tür değişkenlere dinamik değişken adı verilir. Dinamik değişkenlerin bir özelliği, dinamik değişken için ayrılan alanın, yeterince kullanılmasından sonra yok edilmesidir. Böylece aynı bellek bölgeleri, başka amaçlar için tekrar kullanılabilir hale getirilir. Dinamik değişkenlerin tanımlanması veya tanım cümleleri, statik değişkenler kadar açık ve yalın değildir. Dinamik değişkenler sadece birer değişken ismi ile direkt olarak temsil edilemezler. Bunun yerine dinamik değişkenin bellekteki adresini içeren, özel bir değişken kullanılır. Bu özel değişken, pointer değişkendir.

Pointer değişkenler: Dinamik değişkenlerin bellekteki adreslerini gösteren değişkenlere pointer değişken denilir. Diğer bir deyişle, pointer değişkenlerin temsil ettiği veriler başka verileri temsil eden değişkenlerin bellek adresleridir. Pointer değişkenler dinamik değişkenlerin yerlerini göstermekle beraber, kendileri dinamik değişken değildir. Kısacası, pointer değişkenler dinamik değerleri gösteren statik değişkenlerdir.[111] Pointer değişken tanımında bilgi tipi ^, ile belirtilir.

Örneğin; X : ^Integer;

Yukarıdaki örnekte X değişkeninin bir pointer değişkeni olduğu tanımlanır.

---

[111] Bayburan,Bahattin. **Turbo Pascal.** İstanbul, Beta Basım Yayım, 1990, ss. 271-276

Pointerler, ayrıca nesneler için de tanımlanırlar. Nesneye yönelik programlama metodunda nesnelerde tanımlanan pointer değişkenleri şunlardır:

1. Bilgi sahaları gibi tanımlanan pointer göstergeleri.
2. Statik metod ile bir değişken gibi tanımlanan pointer'ler.
3. Bu iki tanımlamanın bir arada kullanılması metodu, Polymorphism (değişik şekilleri olan tanımlama metodu) denilir. Hem statik hem de virtual metod kullanılır. Nesneler farklı tiplerde ve farklı ölçülerde olabilir.

3.2. GELİŞTİRİLEN ÖRNEK ÇİZİM PROJESİ

Turbo Pascal nesneye yönelik üst düzey programlama dili ile geliştirilen, nesneye yönelik çizim projesi, DOS ve WINDOWS kullanıcılarının, her iki ortamda da çeşitli geometrik şekiller çizmelerini, kalem kullanarak el ile bilgisayarda çizim yapmalarını, düz çizgi çizebilmelerini sağlayan, bu şekillerin ve yapılan çizimlerin taşınabilmesine ve gereken yerlerinin silinmesine imkan veren ve bütün bunları istenilen renkte ve büyüklükte gerçekleştirebilen bir projedir. Ayrıca, projede çizimler üzerine, veya kullanım alanı içerisinde herhangi bir yere, belirli fontlarda yazı yazabilmek de mümkündür.

Projede hazırlanan şekiller, ekranda görüntülenebilmekte, diske kayıt edilebilmekte ve yazıcıdan çıktı olarak alınabilmektedir. Proje ile ilgili daha detaylı bilgiler, aşağıdaki kesimde ayrıca açıklanmaktadır:

3.2.1. Geliştirilen Çizim Projesinin İncelenmesi

Bu kesimde, projede kullanılan programlar ve dosyalar ile ilgili kısa bilgiler verilmekte, programların derlenme ve bağlanma yapıları anlatılmakta, ayrıca ana programın akış şeması verilmektedir.

Projede kullanılan programlar, unit'ler, yardımcı programlar ve dosyalar şunlardır. Geliştirilen çizim projesinde, 1 ana program, 4 unit, l5 yardımcı program 20 tuş dosyası, 2 açılış dosyası ve turbo pascalın standart unit'leri bulunmaktadır.

CIZIM.PAS : Ana program.
TUS.PAS : Buton objesini içeren, tus unit'inin bulunduğu dosya.

FKULLAN.PAS: Programda mouse kullanımı için gereken alt programları içeren, fkullan unit'inin bulunduğu dosya.

FKONBEL1.PAS: Bir mouse tıklaması oluşunca, imlecin o anda ekranın neresinde olduğunu bulup, bu yere ilişkin bir durum kodu döndüren, fare konumu belirleme unitini içeren dosya.

PENCERE1.PAS: Ekranda menu ve uyarı pencerelerini açmakta kullanılan fonksiyonları içeren, pencere1 unitin dosyası.

TUS1.PAS: 1 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS2.PAS: 2 nolu tuşun üzerindeki resmi oluşturan yardımcı program

TUS3.PAS: 3 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS4.PAS: 4 nolu tuşun üzerindeki resmi oluşturan yardımcı program

TUS5.PAS: 5 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS6.PAS: 6 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS7.PAS: 7 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS8.PAS: 8 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS9.PAS: 9 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS10.PAS: 10 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS11.PAS: 11 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS12.PAS: 12 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS13.PAS: 13 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS14.PAS: 14,15,16,17,18,19 nolu tuşların üzerlerindeki resimleri oluşturan yardımcı program.

TUS20.PAS: 20 nolu tuşun üzerindeki resmi oluşturan yardımcı program.

TUS1.IMG: 1 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS2.IMG: 2 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS3.IMG: 3 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS4.IMG: 4 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS5.IMG: 5 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS6.IMG: 6 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS7.IMG: 7 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS8.IMG: 8 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS9.IMG: 9 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS10.IMG: 10 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS11.IMG: 11 nolu tuşun üzerindeki resmin saklandığı dosya.

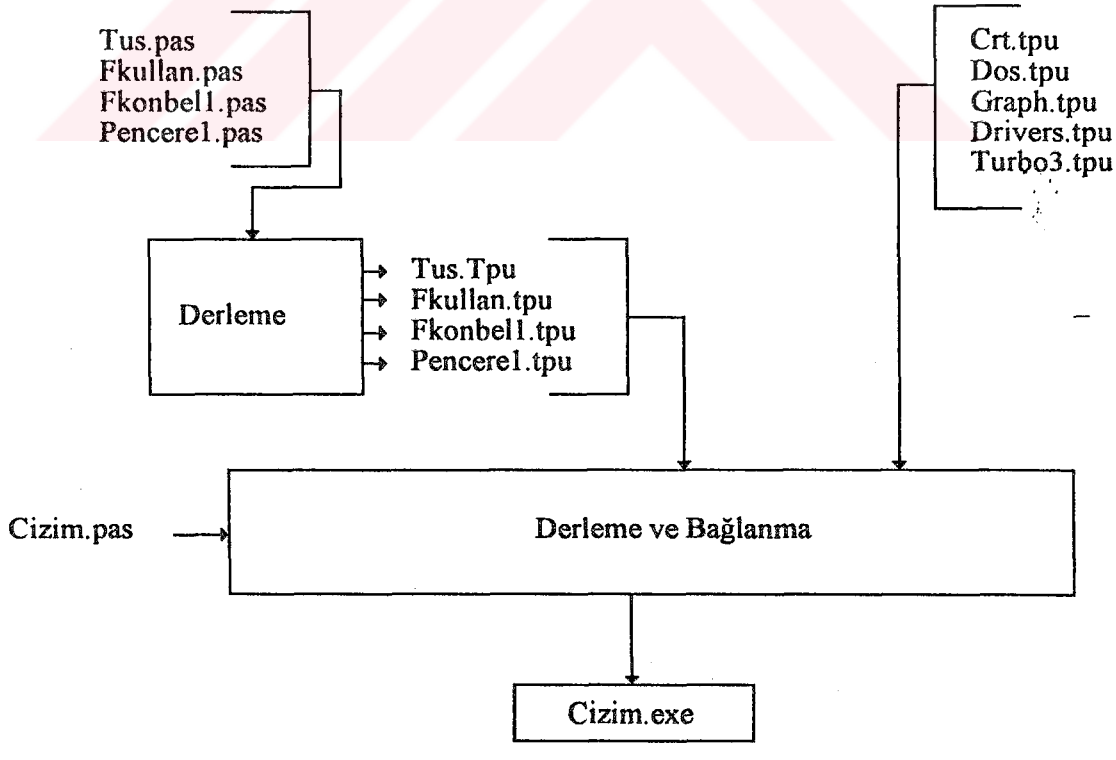
TUS12.IMG: 12 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS13.IMG: 13 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS14.IMG: 14 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS15.IMG: 15 nolu tuşun üzerindeki resmin saklandığı dosya.

TUS16.IMG: 16 nolu tuşun üzerindeki resmin saklandığı dosya.
TUS17.IMG: 17 nolu tuşun üzerindeki resmin saklandığı dosya.
TUS18.IMG: 18 nolu tuşun üzerindeki resmin saklandığı dosya.
TUS19.IMG: 19 nolu tuşun üzerindeki resmin saklandığı dosya.
TUS20.IMG: 20 nolu tuşun üzerindeki resmin saklandığı dosya.
ILKEK.1MG: Açılış ekran görüntüsünün saklandığı dosya.
ILKEK.2MG: Açılış ekran görüntüsünün saklandığı dosya.
*.BGI: Grafik ekran sürücülerini içeren turbo pascal dosyaları.
*.CHR: Karakter setini içeren turbo pascal dosyaları.
CRT.TPU: Ekranla ilgili komutların bulunduğu unit.
DOS.TPU: İşletim sistemiyle ilgili komutlar unit'i.
GRAPH.TPU: Grafik ekranda kullanılan fonksiyon ve prosedürlerin bulunduğu unit dosyası.
TURBO3.TPU: Turbo pascal 3. versiyon farklılıkları unit'i.
TUS.TPU: Tus.Pas unit'inin derlenmiş hali
FKULLAN.TPU: Fkullan.Pas unit'inin derlenmiş hali
FKONBEL1.TPU: Fkonbel1.Pas unit'inin derlenmiş hali
PENCERE1.TPU: Pencere1.Pas unit'inin derlenmiş hali
Programın derlenme ve bağlanma yapısı şekil 3.1'de gösterilmektedir.

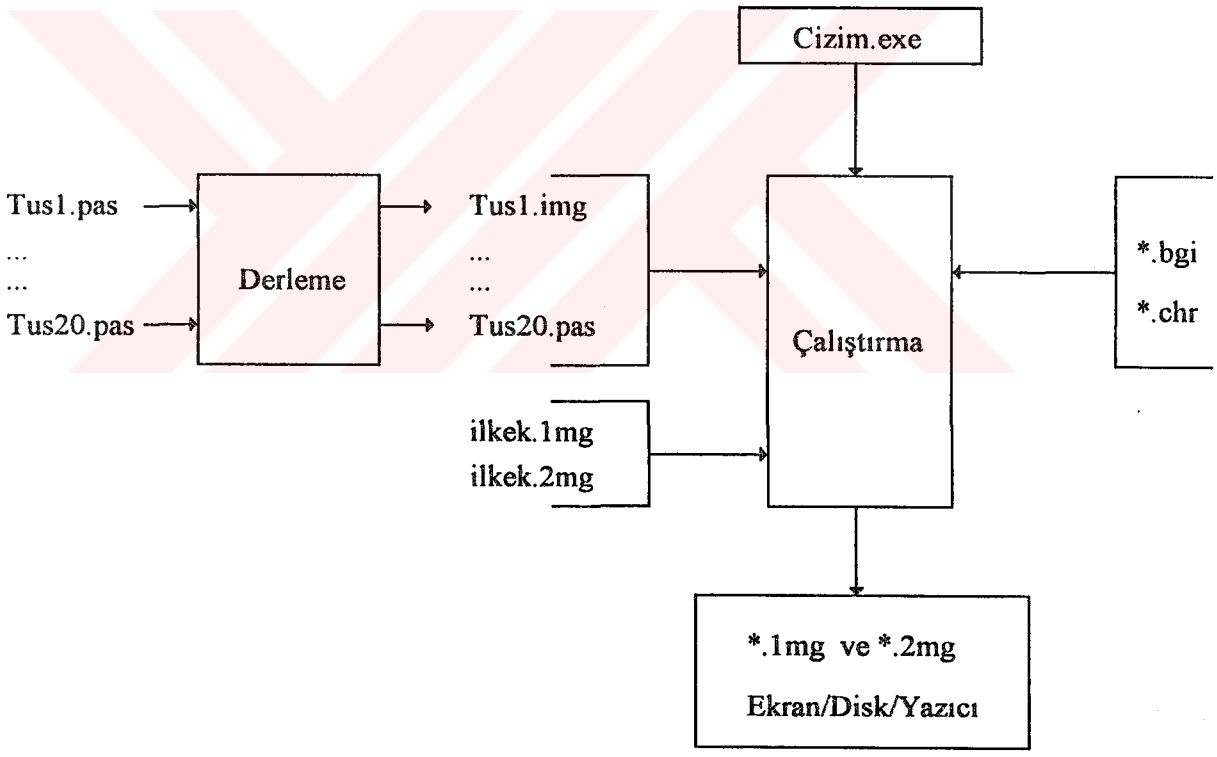


Şekil 3.1. Çizim Programının .EXE Hale Getirilmesi

Programın derlenmesi sırasında, ana program ile birlikte 4 unit ve turbo pascal'ın standart unitleri derleme ve bağlanma yaparak cizim.exe yi oluştururlar. Ancak programın çalışabilmesi için .exe program ile birlikte grafik ekranla ilgili dosyalar, yazı fontlarıyla ilgili dosyalar, projenin ilk ekran görüntüleri ile çalışma alanında kullandığımız tuş görüntülerinin de yüklenebilmesi gerekir. Bu nedenle de söz konusu dosyalar ilgili disk veya dizinde bulunmalıdır.

Şekil 3.2, .EXE hale getirilen programın, çalıştırılma şeklini göstermektedir. Cizim.exe elde edildikten sonra, programın çalıştırılabilmesi için şu dosyalara ihtiyaç duyulmaktadır.

- Tus*.img
- ilkek.1mg ve ilkek.2mg
-*.bgi ve *.chr



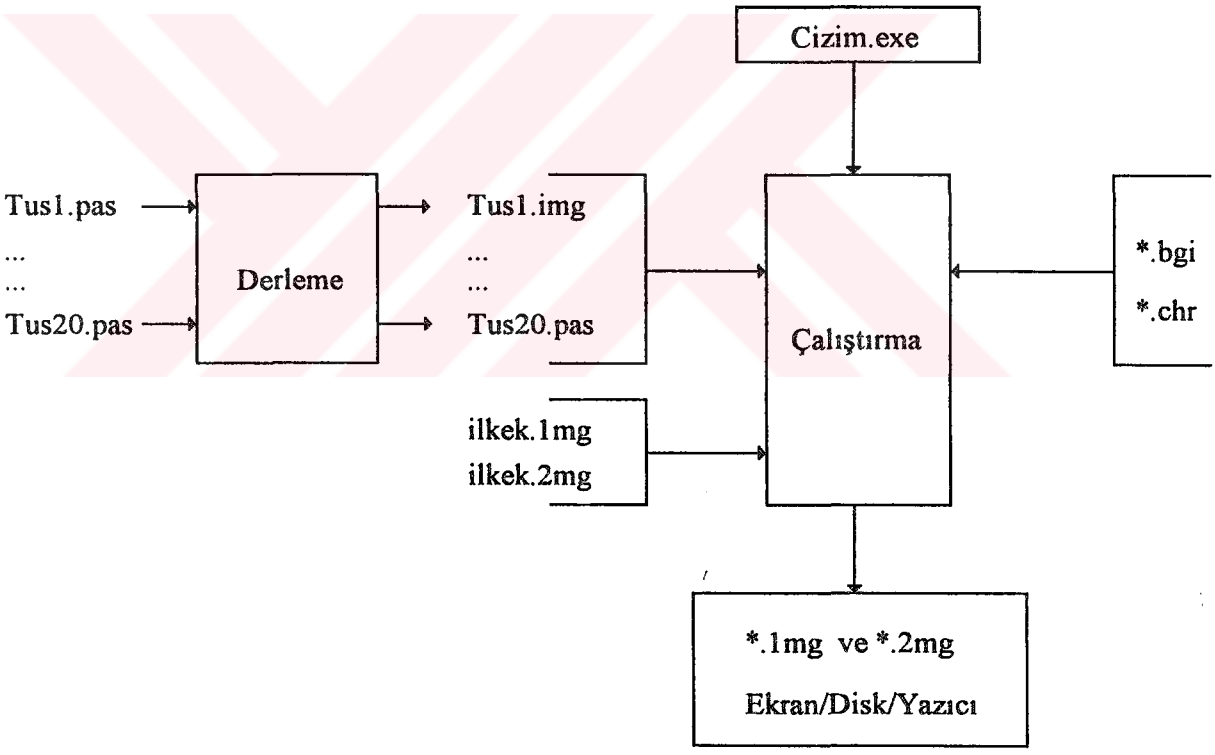
Şekil 3.2. CIZIM.EXE Programının Çalıştırılma Yapısı

Tus.Pas : Bu unit buton objenin tanımlandığı dosyadır. Bu dosyadaki buton objesi şu şekilde tanımlanmaktadır.

Programın derlenmesi sırasında, ana program ile birlikte 4 unit ve turbo pascal'ın standart unitleri derleme ve bağlanma yaparak cizim.exe yi oluştururlar. Ancak programın çalışabilmesi için .exe program ile birlikte grafik ekranla ilgili dosyalar, yazı fontlarıyla ilgili dosyalar, projenin ilk ekran görüntüleri ile çalışma alanında kullandığımız tuş görüntülerinin de yüklenebilmesi gerekir. Bu nedenle de söz konusu dosyalar ilgili disk veya dizinde bulunmalıdır.

Şekil 3.2, .EXE hale getirilen programın, çalıştırılma şeklini göstermektedir. Cizim.exe elde edildikten sonra, programın çalıştırılabilmesi için şu dosyalara ihtiyaç duyulmaktadır.

- Tus*.img
- ilkek.1mg ve ilkek.2mg
-*.bgi ve *.chr



Şekil 3.2. CIZIM.EXE Programının Çalıştırılma Yapısı

Tus.Pas : Bu unit buton objenin tanımlandığı dosyadır. Bu dosyadaki buton objesi şu şekilde tanımlanmaktadır.

```
type buton = object
        numara,
                {Butonun numarası}
        kalınlık,
                {Buton çizilirken ne kadar kabarık çizileceği}
        durum = byte;
                {Butonun basılı (1) veya normal (0) olduğunu
belirtir.}
        butonx1,butony1,butonx2,butony2=integer;
                {Butonun ekrandaki solüst ve sağalt köşeleri}
        desendosyası = string;
                {Buton üzerine yapıştırılacak resim dosyası adı}
        procedure baslat;
                {Butonu normal duruma getirir.}
        procedure ciz;
                {Butonun durum değerine göre, buton
koordinatlarına basılı veya normal durumda bir tuş çizer.}
        procedure donat;
                {Desen dosyasından resmi getirip butonun üzerine
yapıştırır.}
        procedure
led(acıkrenk,kapalırenk:byte;bx,by,ux,uy:integer);
                {Butonun üst köşesinden (bx,by) kadar içerden,
(ux,uy) ye kadar olan dikdörtgende, tuş basılı durumda ise acıkrenk, tus normal
durumda ise kapalırenk değişkeninde gelen renkte bir lamba (led) çizer.}
        procedure resimliçiz(onoff:byte);
                {onoff değişkeninin değerine göre ya basılı ya da
normal durumda, resimli buton çizer; buton numarası 1 ile 9 arasında ise
botonların sol üst köşesine bir led çizer.}
    end;
```

Fkullan.pas : Projede mouse kullanılabilmesi için gereken altprogramları içeren unit dosyasıdır.

Bu altprogramlar şunlardır;

```
Type  farefonk=(goster,gizle,sifirla);
      function Farevar:boolean;
                {Bilgisayara bağlı, sürücüsü o an aktif olan bir
mouse varsa true, yoksa false döndürür.}
```

procedure fare(ffonk:farefonk);

{ffonk değişkeninin değerine göre fare imlecini ekranda görünmez yapar veya sonra tekrar görünür yapar.}

function farex:word;

{Fare imlecinin o anda ekranda bulunduğu noktanın x koordinatını döndürür.}

function farey:word;

{Fare imlecinin o anda ekranda bulunduğu noktanın y koordinatını döndürür.}

function sold:boolean;

{O anda mouse'un sol düğmesine basılı olup olmadığını öğrenmek için kullanılır. (true:basılı, false:basılı değil.}

function sagd:boolean;

{O anda mouse'un sağ düğmesine basılı olup olmadığını öğrenmek için kullanılır. (true:basılı, false:basılı değil.}

function ftusubasılı:boolean;

{O anda mouse'un sağ veya sol düğmesinden herhangi birine basılı olup olmadığını öğrenmek için kullanılır. (true:herhangi bir tusa basılı, false:basılı değil.}

procedure yataysinir(minx,maxx:word);

{fare imlecinin ekran üzerinde hareket bilgisini soldan minx, sağdan maxx'e olan aralıkta sınırlamak için kullanılır.}

procedure dikeysinir(miny,maxy:word);

{fare imlecinin ekran üzerinde hareket bilgisini üstten miny, alttan maxy'ye olan aralıkta sınırlamak için kullanılır.}

Fkonbel1.pas : Bu unit dosyasında, fare imlecinin ekranda konumunu belirleyerek gereken işleri yapan prosedürler bulunur. Ayrıca bu dosyada, o an seçili olan renk (geçerlirenk) ve seçili çizgi kalınlığı (geçerlicizgik) değişkenleri de vardır.

procedure aktifrenk (yenirenk=word);

{Bu prosedür, fare imlec, ekranın yanındaki renk paletinin üzerine getirilip tıklanınca çağrılır ve imlecin konumuna göre paletten hangi rengin seçilmek istendiği belirlenerek yenirenk değişkeni ile bu prosedür gelir. Bu prosedür ise aktif renk kutusunun içini bu renkte doldurur ve geçerlirenk değişkenine de yenirenk değerini atar.}

procedure farekonumubelirle(x,y:integer;var durum:byte;var tusno:byte)

{Bu prosedür, ekran üzerinde herhangi bir yere fare ile tıklayınca çağrılır. Farenin o anki konumu x,y değişkenleri ile bu prosedüre gönderilir. Bu prosedürün görevi x,y koordinatlarına bakarak kullanıcının ne yapmak istediğini (ekranda hangi bölgeyi tıkladığını) anlayıp durum değişkeninde geri döndürmektir.}

Durum değişkeninde döndürülen değerler şunlardır.

1. Hiçbir tuşa, palete ve çalışma bölgesine tıklanmamış. (anlamsız bir yer seçilmiş)
2. Çalışma bölgesi içinde bir noktaya tıklandı
3. Palet üzerinde bir yere tıklandı ve geçerlirenk değişkeni güncellendi
4. Çıkış (SON) tuşuna tıklandı ve tusno=20
5 Çizim (Fonksiyon) tuşlarından herhangi birine tıklandı ve tusno=1 ... 8
6. Ayar tuşlarından herhangi birine tıklandı ve tusno=9 ...13
7. Dosya tuşlarından herhangi birine tıklandı ve tusno=14 ... 19

Pencere1.pas : Ekrana sorgulama menüsü ve dosya okuma penceresi için, çerçeveli bir pencere oluşturulması işlerini yürütecek fonksiyonları içeren, unit dosyası.

Bu dosyada bulunan altprogramlar ise şunlardır:

var p: pointer;bellek:integer;

function

pencereac(baslık:string;x1,y1,x2,y2:integer):integer;

{Sol üst köşesi (x1,y1), sağ alt köşesi (x2,y2) noktaları ile verilen dikdörtgen bölgeye çerçeveli bir pencere çizer. Pencerenin ilk kısmına ise başlık değişkeni ile gelen yazıyı yazar. Pencereyi çizmeden önce o yerde bulunan şekilleri hafızaya alır p pointerinin gösterdiği yere saklar. Saklanan bölgenin hafızada kapladığı alanın boyu pencereac fonksiyonunun geri dönüş değeri olur. Bu değer daha sonra pencerekapat içinde gerekli olacak.}

procedure pencerekapat(x1,y1,boy:integer);

{Daha önce p, pointer'inin gösterdiği yere kayıt edilmiş pencerenin altındaki şekillerin, (x1,y1) ile verilen nokta, sol üst köşesi olacak şekilde ekrana tekrar getirilip yapıştırılması işini yapar;boy geğişkeni ile belirtilen miktarda belleği de serbest bırakır. Boy saklanmış alanın bellekte kapladığı yerin uzunluğudur ve pencereaç fonksiyonunun geri dönüş değeridir.}

Çizim.pas :(Ana Program)

Ana program, 26 prosedür, 1 fonksiyon içermektedir. En önemli prosedür, baskatus prosedürüdür ve programın çatısını oluşturur. Ana programda tanımlanan önemli sabit ve değişkenler şunlardır.

```
fonts : array[0..10] of string[11] =
(' Default ',' Triplex ',' Small  ',' SansSerif ',' Gothic ',  ' Script  ',' Simplex ', 'Trip.Script',' Complex ',  ' European ',' Bold  ');
        {Yazı modunda kullanılabilecek fontların isimler listesi.}
tuslar:array[1..20] of buton;
        {programdaki 20 adet tus buton objesi tipinde tanımlanıyor.}
gecerliharfbuyuk:byte;
        {Ayar tuşlarıyla, yazı yazma tuşuna basılınca yazılacak harflerin büyüklük değerini tutmak için kullanılan global bir değişken.}
aktifyazitipino :byte;
        {Yazı modunda hangi karakter kümesinin kullanılacağı bilgisini tutan global değişken.}
acıkdosyadegisti:boolean;
        {Çalışma bölgesinde şu anda açık bulunan çizimin en son halinin bir dosyada kayıtlı olup olmadığı bilgisini tutar.}
advar:boolean;
        {Çalışma bölgesinde şu anda açık bulunan çizimin bir isminin olup olmadığını belirten global değişken.}
dosyaadı:string;
        {Çizimleri dosyaya saklamak veya dosyadan getirmek için kullanılacak ismin tutulduğu değişken.}
```

Ana programda kullanılan fonksiyon ve prosedürler ise aşağıdaki gibidir.

```
procedure grafik_kur;
        {Grafik sürücünün yüklenmesi işini yapan prosedüre. Grafik sürücü dosyasını bulamaz ise program çalışmaz ve mesaj vererek durur.}
procedure faretest;
        {Mouse sürücüsünün kurulu olup olmadığını test eder.}
procedure ilkekran;
        {program çalışmaya başlayınca ilk görüntüyü ekrana getirir. Görüntü iki parça halinde ilkek.1mg ve ilkek.2mg dosyalarından okunarak getirilir.}
procedure ekran_zemini;
        {Çalışma bölgesini ve etrafındaki gri zemini oluşturur.}
procedure renkler;
        {Soldaki renk paletini oluşturur, aktifrenk çerçevesini çizer ve ilk seçili renk olarak siyah ayarlar.}
```

function onay (yazı:string):boolean;

{Ekranın ortasına, yazı değişkeninde gelen text'in başlık olarak yeraldığı sorgu penceresini açar. Fare ile tıklanarak seçim yapılmasına imkan verir. Seçim yapılınca pencere kapatılır. Evet seçilmişse true, hayır seçilmişse false değeri döndürür. Pencereac ve pencerekapat fonksiyonlarını çağırır.

procedure dosyaadı_oku(var ad:string;var harfsayısı:byte);

{Ekranın ortasında dosya ismi okuma penceresi açar ve bir dosya ismi girilmesi imkanını sağlar. Okunan dosya ismi ad değişkeninde, kaç harfli olduğu ise harfsayısı değişkeninde geri gelir. Eğer vazgeçilecekse (Esc) tuşuna basılır ve ad[0]'a, #9 değeri atanır.}

procedure ekransakla(dad:string);

{Çalışma bölgesini, iki parça halinde dad değişkeni ile verilen isimle, *.1mg ve *.2mg uzantılı dosyalar kaydeder.}

procedure ekrangetir(dad:string;var sonuc:boolean);

{dad değişkeni ile verilen isimde saklanmış görüntü dosyalarını getirir ve çalışma bölgesine yapıştırır. Dosya bulunamazsa sonuç false'dir.}

procedure yenisayfa;

{Çizim alanını boşaltır. Boşaltmak için çizim alanında o anda var olan şeklin güncel kopyasının dosya olarak bulunup bulunmadığına bakılır. Eğer yok ise, yani çizimin güncel hali sadece ekranda ise bir dosyaya kayıt etmemiz gerekir. Daha önceden bu çizimle ilgili bir dosya adı var ise mesele yok, dosya adı yok ise dosyaadı_oku prosedürü ile bir dosya adı okunarak, çizim o isimdeki dosyaya kaydedilir. En son iş olarak da çalışma bölgesi temizlenir.}

procedure dosyaac;

{Çizim alanında, bir dosya ismi okuyarak bu dosyadaki resmi getirir. Ama daha önce çizim alanında herhangi bir resim var ise onu saklamamız gerekir. Bu durumda da yine çizimin bir adı yok ise dosyaadı_oku prosedürü ile bir dosya adı okuyarak ekran bilgilerini dosyaya kaydediyoruz.}

procedure kaydet;

{Onay aldıktan sonra çizim alanındaki şekli bir dosyaya kaydeder. Daha önceden bu çizimle ilgili bir isim verilmemişse dosyaadı_oku prosedürü ile bir dosya ismi okur sonra kaydeder.}

procedure yeniadver;

{Çizim alanındaki resmi, onay alarak ve yeni bir dosya ismi okuayarak kaydeder.}

procedure yazıcı;

{Onay aldıktan sonra, çalışma alanı içerisindeki resmi printer'den yazdırır. Yazdırmak için 5 numaralı interrupt kullanılır.}

procedure tus_koordinatlarını_ata;

{Tuşlar adlı buton dizisinin her bir elemanına ait koordinatlara ve buton üzerindeki resmin bulunduğu dosyanın ismine (desen dosyası), ilk değer atanır.}

procedue altpencereler;

{Çizgi kalınlığı, harf büyüklüğü ve harf tipi ayar düğmelerinin etrafındaki çerçeveleri çizer ve isimlerini yazar.}

procedure aktifyazıtipi {islem:byte);

{islem=0 ise aktifyazıtipino değişkeni bir azaltılır, işlem=1 ise bir artırılır ve yazı stilini fonts dizisinin font dizisinin aktifyazıtipino indisli elemanına çevirir.}

procedure yazıtipiyaz;

{Font dizisinin aktifyazıtipino indisli elemanı, 18 ve 19 nolu ayar tusları arasına yazılır.}

procedure cizkalınyaz;

{Gecerlicizgik değişkeninin değerini 14 ve 15 nolu ayar tuşları arasına yazar.}

procedure harfbuyukyaz;

{Gcerliharfbuyuk değişkeninin değerini 16 ve 17 nolu ayar tuşları arasına yazar.}

procedure baskatus(tno:byte);

{Programın en temel prosedürüdür. Tno değişkeni ile hangi tuşa basılmiş ise, gelen o tus değerine göre ilgili prosedürü çağırır. Önce tno nolu tuşu basılmış halde çizer, sonra o tuşa ait prosedürü çağırır. Bu prosedürden dönülünce tuşu tekrar normal halde çizer.}

procedure sakla;

procedure sakla1;

procedure olustur;

{Bu prosedürler, çalışma bölgesinde, bir ucu sabit tutulup diğer ucu mouse ile sürüklenerek çizgi,elips,dikdörtgen çizilirken çizgilerin altında kalan şekillerin bozulmaması için, bu şekilleri belleğe alıp sonra tekrar bellekten okuyarak ekrana yapıştırma işlemini yaparlar.}

procedure kalem;

{Kalem adlı tuşa basıldığında çağrılır. Çalışma bölgesinde mouse tuşu basılı tutulduğu sürece geçerlirenkte ve geçerli çizgi kalınlığında karalamalar yapılabilir.}

procedure cizgi;

{Çizgi adlı tuşa basılınca çağrılır. Çalışma bölgesinde düz çizgiler çizmek için kullanılır. Bu çizgiler geçerlicizgik ve gecerlirenk değişkenleri dikkate alınarak çizilir.}

procedure yay;

{Elips tuşuna basılınca çağrılır. Çalışma bölgesinde mouse ile belirlenecek farklı büyüklüklerde elipsler çizmek için kullanılır. Bu elipsler geçerli renk ve geçerli çizgi kalınlığında çizilirler. Mouse'nin sol buton'una basıldığında içi boş, sağ buton'una basıldığında ise içi dolu elipsler elde edilir.}

procedure dortgen;

{Dörtgen tuşuna basılınca çağrılır. Yay prosedürü ile aynı mantıkla çalışır. Sadece elips yerine dörtgen çizer.}

procedure zikzak;

{Çalışma bölgesinde, sadece bir kere tıklanarak bitişik çizgiler çizilmesini sağlar.}

procedure kestası;

{Çalışma bölgesi içerisinde taşınmak istenen herhangi bir bölge önce bir dörtgen çizilerek işaretlenir, sonra fare ile sürüklenerek başka bir konuma bırakılır.}

procedure silgi;

{Silme tuşuna basılarak aktif duruma getirilir. Fare butonuna basıldığı sürece, çalışma alanı içerisinde mouse'nin üzerinden geçtiği bölgelerin silinmesini sağlar. Silginin büyüklüğü, gecerlicizgik değişkeninin aldığı değer ile orantılıdır.}

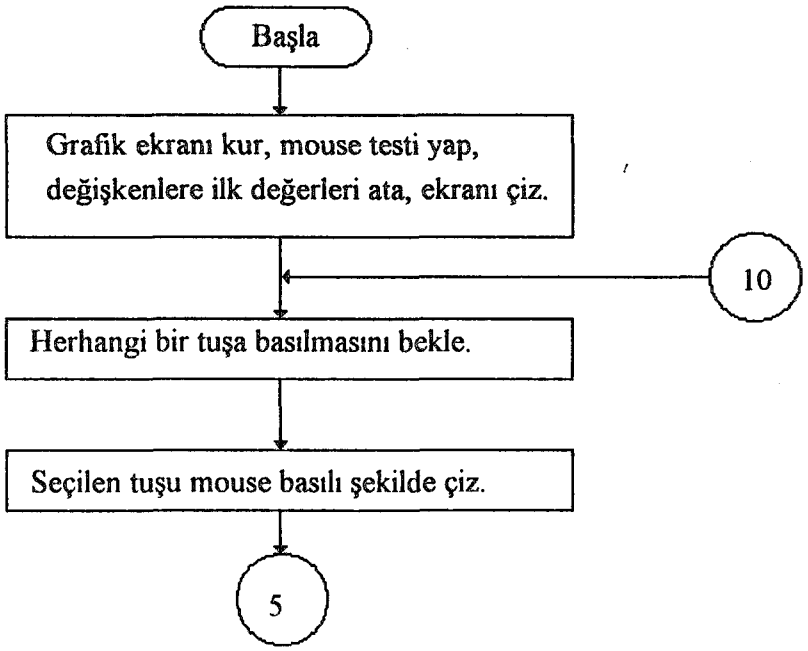
procedure yazı;

{Aktifyazıtipino ve gecerliharfbuyuk değişkenleri dikkate alınarak çalışma bölgesinde mouse ile işaretlenen herhangi bir konuma klavye ile yazı yazılmasını sağlar.}

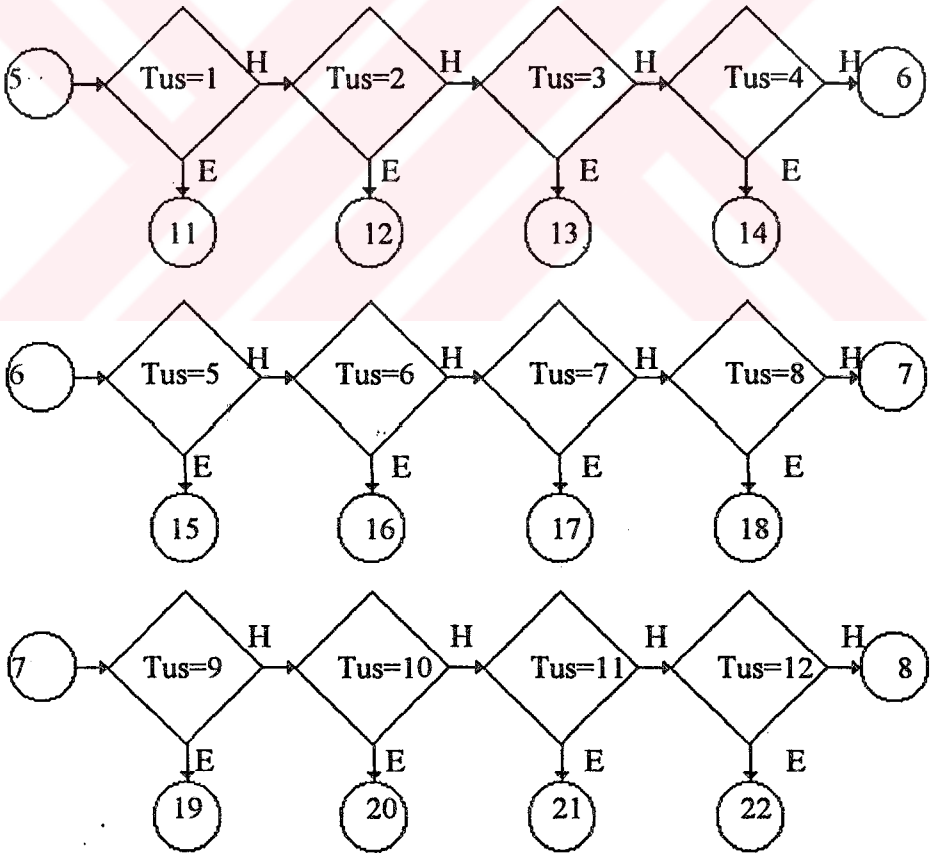
Ana program icra bloğu:

{Gerekli ilk değer atamaları ve kontrolleri yapıldıktan sonra bir sonsuz çevrime girer. Bu çevrim içerisinde ilgili tuslardan herhangi birine basıldığında baskatus prosedürü çağrılır. Tuş numarasına göre ilgili prosedürler işlem görür. Bu çevrimden tek çıkış yolu, 20 nolu tuşa basılmasıdır. Bu tuşa basılınca, program sona erer.}

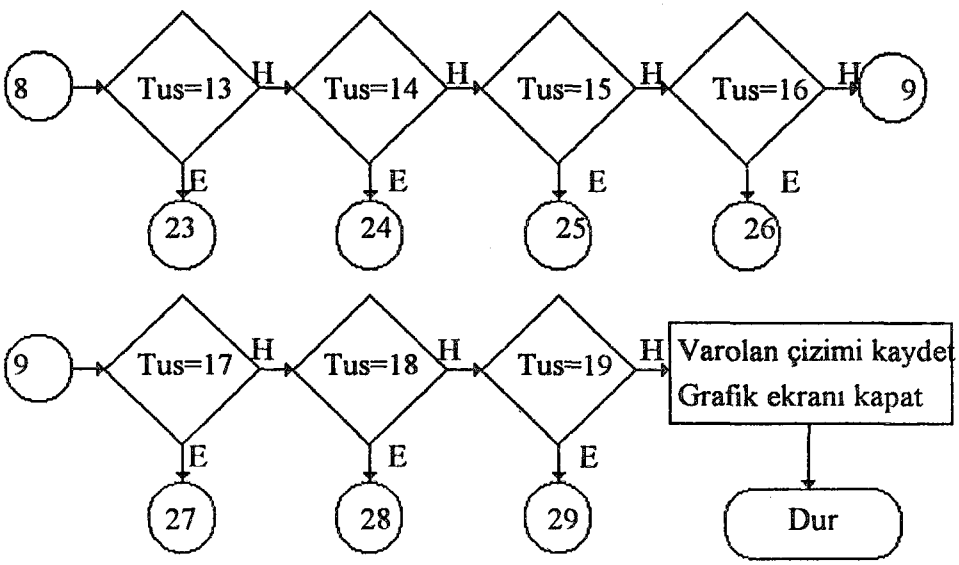
Ana programın akış şeması şekil 3.3'de gösterilmektedir.



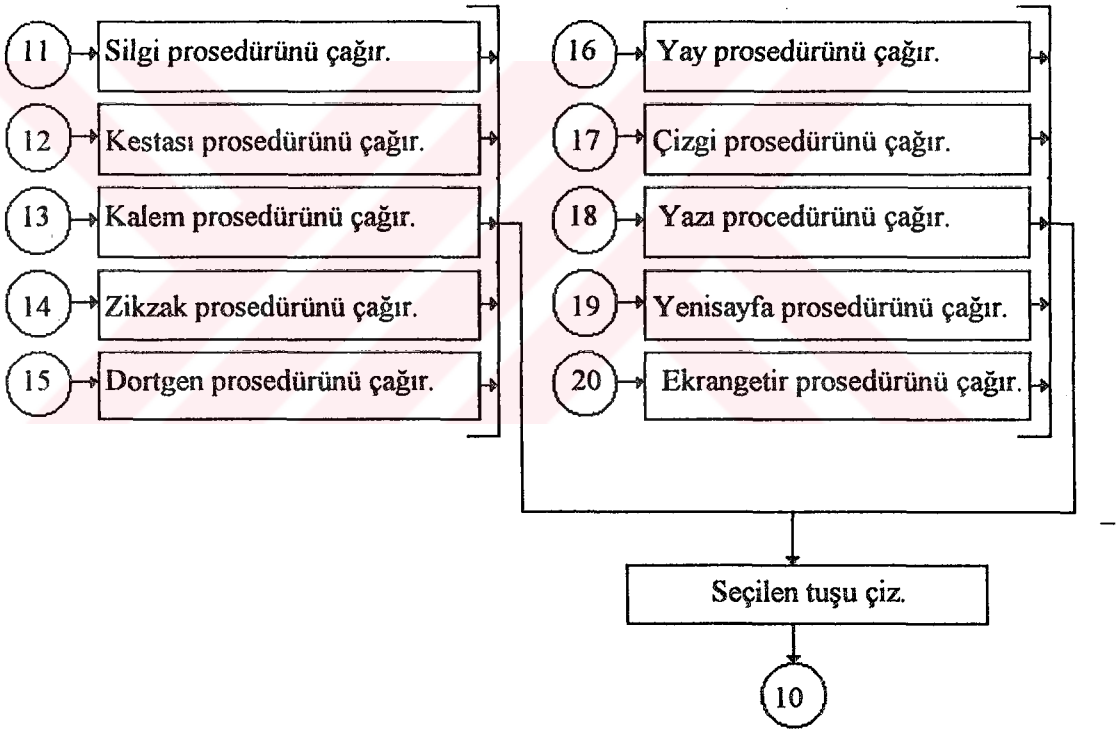
Şekil 3.3. Ana Program Akış Şeması (Devam Ediyor)



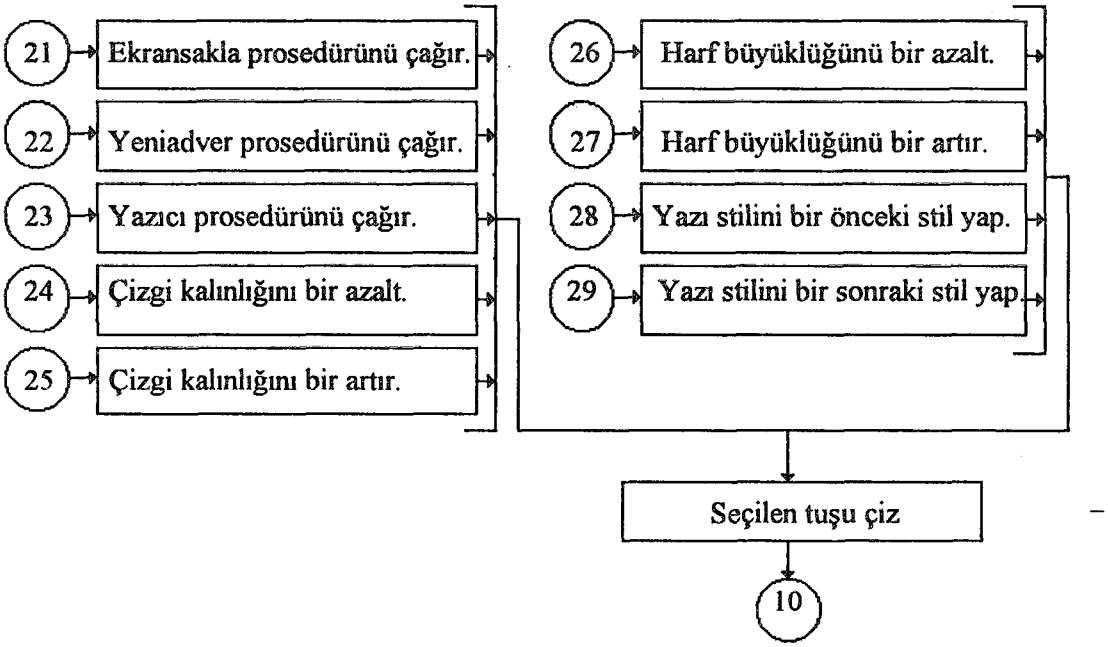
Şekil 3.3. Ana Program Akış Şeması (Devam Ediyor)



Şekil 3.3. Ana Program Akış Şeması (Devam Ediyor)



Şekil 3.3. Ana Program Akış Şeması (Devam Ediyor)



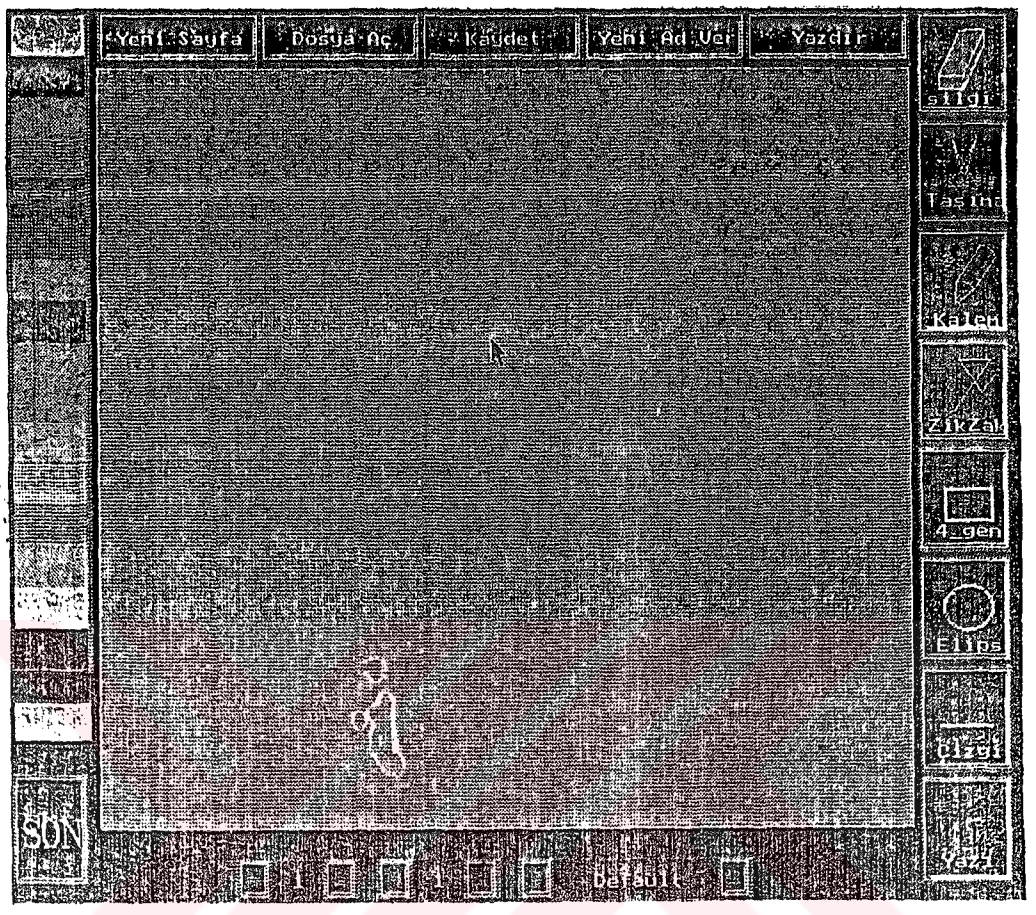
Şekil 3.3. Ana Program Akış Şeması

3.2.2. Geliştirilen Örnek Uygulamanın Çalıştırılması

Geliştirilen çizim programı, disketten veya kayıtlı bulunduğu sürücüden proje adı olan CIZIM yazılarak, çalıştırılabilir. Programın çalışabilmesi için, *.bgi ve *.chr dosyaları ile Tus1.img'den Tus20.img'ye kadar olan tuş dosyalarının, ayrıca ilk ekranla ilgili olarak ilkek.1mg ve ilkek.2mg dosyalarının aynı sürücüde bulunmaları gerekir. Özellikle, grafik sürücüyle ve mouse ile ilgili dosyaların mutlaka bulunması lazımdır.

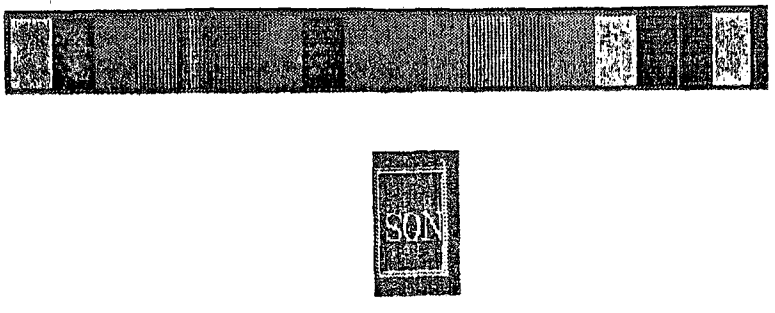
CIZIM komutu girildiğinde, ilk ekran görüntülenir. Asıl çalışma ortamına, buradan, Enter, tuşuna basılarak geçilebilir. Çalışma ekranı görüntülendiğinde, ekranda dakkati çeken önemli özellikler şunlardır:

1. Şekil 3.4'de görüntülenen, ekranın ortasındaki gri çerçeve içerisinde bulunan açık yeşil alan, çalışma alanıdır.



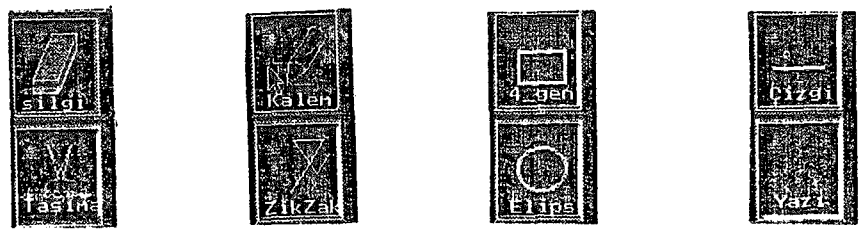
Şekil 3.4. Çalışma Alanı

2. Ekranın sol kenarında, renk seçenekleri mevcuttur. Aktif renk, siyah olarak belirlenmiştir. Ayrıca yine, ekranın sol alt köşesinde SON (çıkış) tuşu, bulunmaktadır. Renk seçenekleri ve çıkış tuşu, şekil 3.5'de verilmektedir.



Şekil 3.5. Renk Seçenekleri ve Çıkış Tuşu

3. Ekranın sağ tarafında 8 adet fonksiyon tuşu vardır. Bunlar sırasıyla; Silgi, Kesme-Taşıma, Kalem, Zikzak, Dörtgen(içi dolu veya boş), Elips(içi dolu veya boş), Çizgi ve Yazı tuşlarıdır. Bu tuşlar şekil 3.6'da görüntülenmektedir.



Şekil 3.6. Fonksiyon Tuşları

4. Ekranın üst kısmında 5 adet tuş bulunmaktadır. Bunlar yeni çalışma ortamının açılması, daha önce oluşturulmuş dosyaların görüntülenmesi, çalışılan andaki bilgilerin kaydedilmesi, herhangi eski bir dosyanın veya üzerinde çalışılan bilgilerin farklı bir isimle yeniden oluşturulması ve ekrandaki bilgilerin yazıcıdan alınması işlemlerini gerçekleştiren tuşlardır. Şekil 3.7'de, bu tuşlar gösterilmektedir.

Şekil 3.7. Kayıt ve Yazdırma Tuşları

5. Ekranın alt kısmında ise şekil 3.8'de görüntülendiği gibi çizgi kalınlığını artıran ve azaltan, harf büyüklüğünü artıran ve azaltan tuşlar ile harf tipini ileri ve geri değiştiren tuşlar bulunmaktadır.

Şekil 3.8. Ayar Tuşları

Bütün bu özelliklerden, mouse kullanılarak kullanıcının isteğine göre yararlanmak mümkündür. Çalışma alanına ilk geçildiğinde, mouse göstergesi ekranın sol üst köşesinde konumlanmış olarak durur. Kullanılmak istenilen özelliğe göre, gerekli tuşun seçilmesi için, mouse'un sol veya sağ tuşlarından

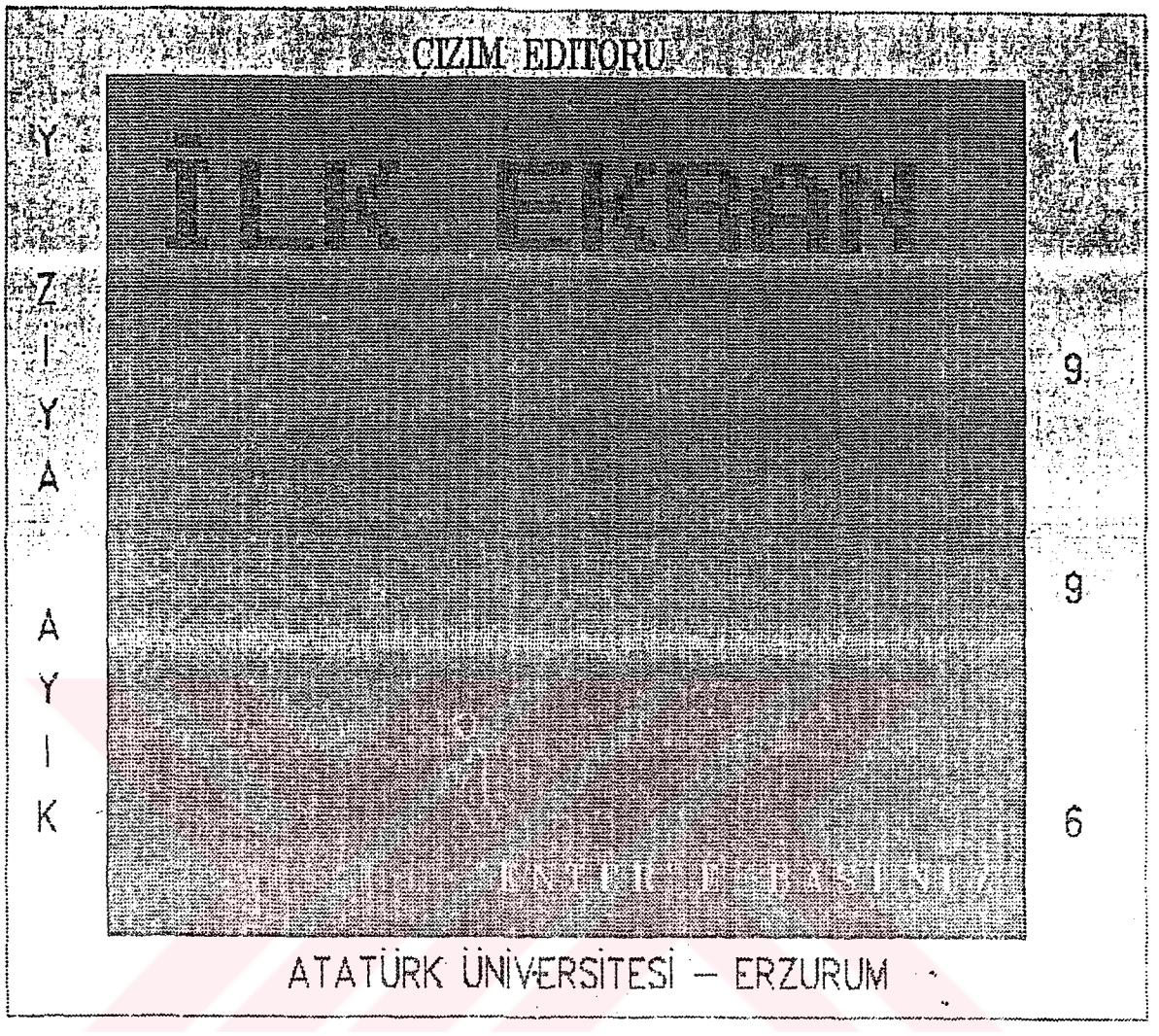
herhangi birine basılmalıdır. Başlangıçta hiçbir tuş seçili değildir. Tuş seçiminde öncelik sırası yoktur. Fonksiyon tuşlarından herhangi biri seçilebilir, istenilen renk, çizgi, harf büyüklüğü ve yazı türü mouse kullanılmak suretiyle belirlenebilir. Her tuş, uzerinde yazılı ve çizili fonksiyonu icra eder.

Çizim alanında, gerçekleştirilen çizimlerin herhangi bir dosyada saklanmak istenmesi halinde, ekranın üst kısmında bulunan tuşlardan Kaydet tuşu, seçilmelidir. Bu durumda, program tarafından bu istek onaylatıldıktan sonra, dosya ismi istenecektir. Ayrıca aynı bilgileri baska bir dosyada saklamak amacıyla da, Yeni Ad Ver tuşu seçilmelidir. Daha önceden oluşturulan dosyaları ekranda görüntülemek için, Dosya aç tuşu, bilgilerin yazıcıdan alınması için, Yazdır tuşu kullanılmalıdır. Yeni çalışma ortamı oluşturmak için de, Yeni Sayfa tuşu seçilmelidir.

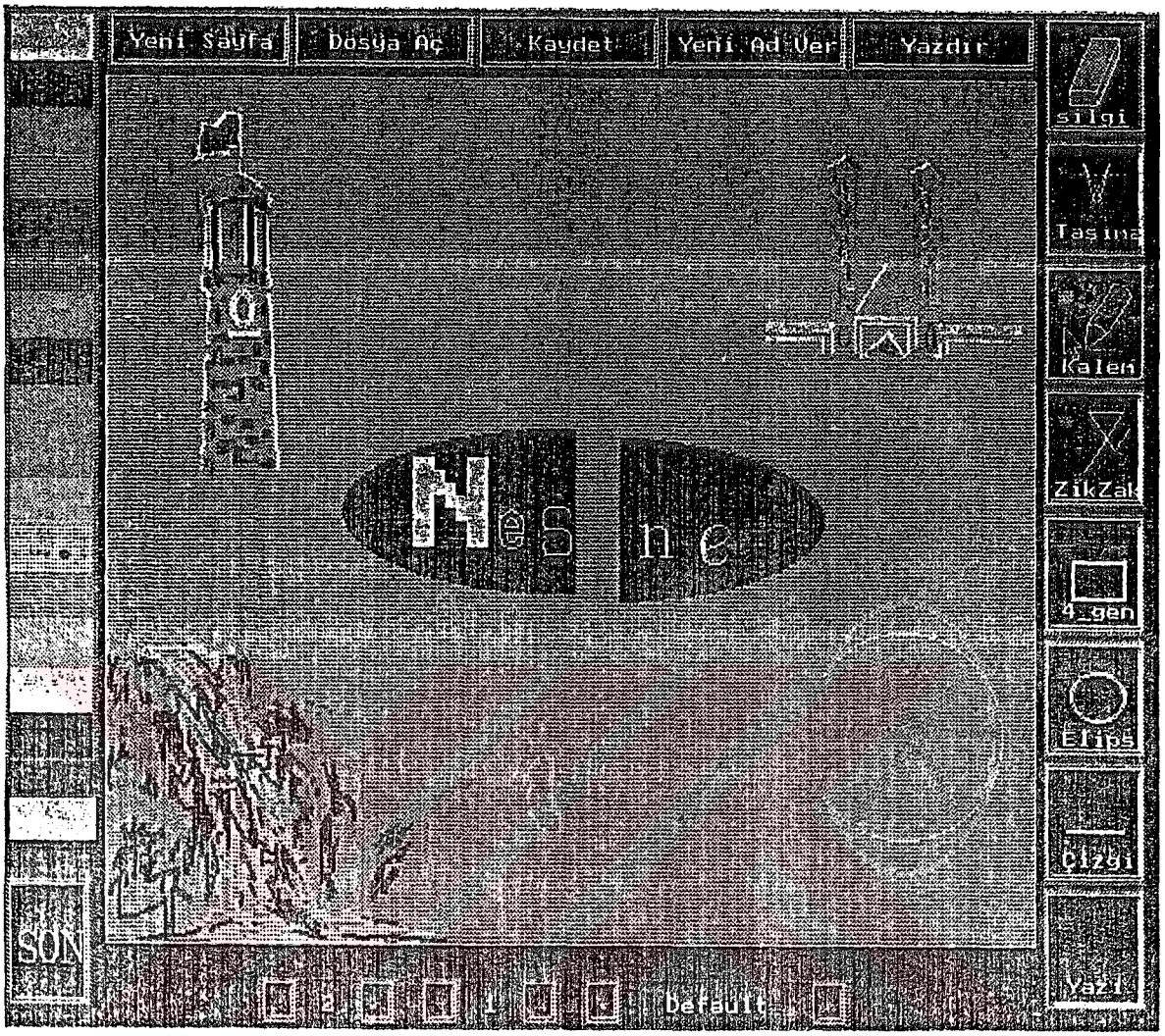
Çalışma esnasında, çizimlerin veya yazıların büyüklükleri ve türleri ekranın altındaki tuşlarla sağlanmaktadır. Her grubun sol tarafı azaltma, sağ tarafı ise artırma seçeneğidir.

İşlemler tamamlandıktan sonra, sol alt köşedeki Son tuşuna basılarak programdan ve grafik ortamdan çıkılır. Bunun haricinde herhangi bir tuşa basılarak, programdan çıkılaması mümkün değildir.

Programın ilk ekranı, ve örnek çalışma ortamı, yazıcıdan alınarak, şekil 3.9, ve 3.10'da verilmektedir:



Şekil 3.9. İlk Ekran



Şekil 3.10. Örnek Çalışma Ortamı

SONUÇ

Programlamanın, insanoğlunun hayatında önemli bir yer tutmaya başlamasından itibaren, birçok programlama teknikleri geliştirilmiştir. Bu teknikler her seferinde, biraz daha kolay öğrenilebilir ve kolay yazılım geliştirilebilir programlama yöntemlerini hedeflemişlerdir. Bilgisayarlardan, daha fazla nasıl ve hangi konularda yararlanılabilir düşüncesi ile geliştirilen yeni teknikler sayesinde, kullanım alanları genişlemiştir. Bilgisayarların tarihsel gelişimleri araştırıldığında, yazılıma paralel olarak, donanım alanında da çok büyük gelişmelerin olduğu gözlenmektedir.

Eski programlama tekniklerinin artık yeterli olamamaları sonucunda, 1980'li yılların başında, nesneye yönelik yaklaşım tekniği kendini kabul ettirmeye başlamıştır. Nesneye yönelik yaklaşımın ilgi çekmeye başlamasından sonra, programlama dillerinde bazı değişiklikler yapılarak nesne etiketi takılmaya çalışılmış, bunun yanısıra tamamen nesneye yönelik, nesne tabanını esas alan yeni programlama dilleri de bu kavramla birlikte olgunlaşmıştır.

Nesneye yönelik yaklaşım tekniği, gerçek dünya nesnelerini örnek alır ve programlama mantığını nesne ve özelliklerini dikkate alarak oluşturur. Bu tekniğin en temel kavramı, nesnedir (obje). Veriler ile onlara ait özellikler, birlikte nesneyi temsil ederler. Ayrıca, bu tekniği oluşturan diğer üstün kavramlar kullanılarak, hem yazılım üretiminin bölümler halinde yapılabilmesi, hem üretilen yazılımlara ihtiyaca göre ilaveler yapılabilmesi ve hem de yazılımın, üretildiği donanımdan bağımsız olarak çalıştırılabilmesi mümkündür.

Bu kavramlardan, sınıflama (class) sayesinde, nesnelerin tek başlarına hareketleri önlenmekte, aynı özellikteki nesnelerin gruplar halinde hareket etmeleri sağlanmaktadır. Ayrıca sınıfların birbirlerini oluşturmaları, birbirlerinin özelliklerini kullanabilmeleri de, kalıtım (inheritance) kavramı ile gerçekleştirilmektedir. Kalıtım, aynı zamanda sınıflar arasındaki hiyerarşiyi de sağlamaktadır. Değişkenler, nesneler ve fonksiyonların aynı sınıfta toplanabilmelerini paketleme (encapsulation), değişik yapılara sahip fonksiyonların aynı isimle kullanılmalarını da çokşekillilik (polymorphism) kavramları gerçekleştirmektedir. Nesneye yönelimin bir diğer temel özelliği de geç bağdaştırma (late binding) kavramı ile oluşmaktadır. Bu sayede, bir programın, hangi nesne üzerinde hangi yöntemi kullanacağı, çalışma anında

oluşmaktadır. Bu durum nesneye yönelimin en büyük avantajlarındandır. Bu sayede esneklik ve taşınabilirlik artmakta, kullanım kolaylaşmaktadır. Nesneye yönelimin bir diğer özelliği de, problemin temel bir grup sahalara bölünmesi ve bölümler üzerinde bağımsız olarak çalışılmasını sağlayan soyutlama (abstraction) kavramıdır. Bu sayede, problemler anlamlı ve orantılı bir şekilde parçalara ayrılarak, çözümlenmektedir.

Saydığımız tekniklerin avantajları nedeniyle, nesneye yönelimin kullanımı her geçen gün biraz daha artmaktadır. Ancak bu yöntemle yazılım geliştirilirken, bazı problemlerle de karşılaşmak mümkündür. Kullanılan programlama tekniklerinde analiz ve tasarım safhalarının yeterince ayrı düşünülememesi, nesneye yönelimin takım çalışmasını gerektirmesi, gerçek dünya ile uyumlu nesnelerin bulunma zorlukları ve geleneksel çalışmalara uygun olmadığı için geleneksel model ile gerçekleştirilen yazılımlar üzerinde nesneye yönelik çalışma yapılamaması gibi önemli problemler nesneye yönelik yaklaşımın kullanımını güçleştiren faktörlerdir. Bu nedenle çok basit programların üretiminde ve fazla değişiklik gerektirmeyecek projelerin hazırlanmasında, bu tekniğin kullanılmasına gerek olmadığı, daha kompleks ve uzun vadeli projelerde bu teknikten yararlanılması gerektiğini düşünmekteyiz.

Tez çalışmasında, nesneye yönelik yaklaşım esas alınarak, nesneyi taban kabul eden, modüler, taşınabilir, hem Dos hem de Windows ortamlarında çalışabilen bir çizim editörü, Turbo Pascal nesneye yönelik programlama dili kullanılarak geliştirilmiştir. Geliştirilen editörde buton objesi tanımlanarak, bu obje'ye dayalı 20 adet buton oluşturulmuş ve bu butonların özellikleri doğrultusunda işlemler gerçekleştirilmiştir. Ayrıca mouse kullanımını gerçekleştiren, mouse konumunu belirleyen, ekrandaki pencereleme sistemlerini ve tuşları yöneten bağımsız unit programlar oluşturulmuştur. Bu unit'ler bir başka projede kullanılabilir durumdadır. Yeniden yazılmalarına gerek yoktur. Çizim editörü projesinde istenilen kalınlıkta ve renkte geometrik şekillerin çizilmesi, kalem kullanılması, çizimlerin ekranda herhangi bir yere taşınması veya silinmesi gerçekleştirilmekte buna ilaveten geometrik ortamda farklı fontlarda ve büyüklüklerde yazı da yazılabilmektedir. Elde edilen çizimler veya şekiller istenilen bir isimle dosya halinde disklerde saklanabilmekte, istenildiğinde ekranda yeniden görüntülenebilmekte ve gerektiğinde yazıcıdan kağıda dökülebilmektedir. Proje, gelişime açıktır ve yeni versiyonları geliştirilebilir.

KAYNAKLAR

Addyman, A. **Specification for the Computer Programming Language Pascal.** New York, ISO/DP, 1983.

Aho, A. V. Sethi, R. ve Ullman, J. D. **Compilers: Principles, Techniques, and Tools.** New York, Addison-Wesley, 1986.

Aho, V. A. ve Ullman, D.J. **Principles of Compiler Design.** New York, Addison-Wesley, 1979.

Altan, Naci. **Turbo Pascal 7.0.** İstanbul, Alfa Basım Yayım, 1995.

Atkinson, Colin. **Object Oriented Reuse Concurrency and Distribution.** Newyork, ACM Press, 1991.

Bancilhon, F. Delobel, C. ve Kanellakis, P. **Building an Object Oriented Database System The Story Of O2.** USA, Morgon Kaufmann Publishers, 1992.

Barstow, D. Shrobe, H. ve Sandewall, E. **Interactive Programming Environments.** New York, McGraw-Hill, 1984.

Bayburan, Bahattin. **Turbo Pascal.** İstanbul, Beta Basım Yayım, 1990.

Besant, C, B. ve Lui, C, W, K. **Computer-Aided Design and Manufacture.** New York, Ellis Horwood Limited, 1986.

Birchenough, A. ve Cameron, J, R. "JSD and Object-Oriented Design". **7th ada UK Conference**. New York. 1989.

Blair, G. Gallagher, J. Hutchisaon, D. ve Shepherd, D. **Object Oriented Languages Systems and Applications**. Newyork, Halsted Press, 1988.

Boehm, B. "Software Engineering Education: Some Industry Needs". **in Software Engineering Education: Needs and Objectives.** Berlin, P.Freemand and A. Wasserman, 1976.

Booch, G. "Object-Oriented Development". **IEEE Transactions on Softwar Engineering.** U.S.A., IEEE, 1986.

Booch, Grady. **Object Oriented Design.** Redwood City, Benjamin/Cummings, 1991.

Borland, D. **Pascal:The Language and Its Implementation.** New York, Wiley, 1981.

Borland. **Turbo Pascal User's Guide.** USA, Borland International, 1990.

Brownston, L. **Programming Expert Systems in OPS5.** New York, Addison-Wesley, 1985.

Bulucu, Feza. **Etkin Modem Haberleşmesi Sağlayan Nesneye Dayalı Bir Yazılım Tasarımı.** İstanbul, Basılmamış Y.Lisans Tezi, 1991.

Caruso, M. ve Sciore, E. **The Vision Object-Oriented Database Management System.** Newyork, ACM Press, 1987.

Coleman, C. Arnold, P. ve Bodoff, S. **Object Oriented Development The Fusion Method.** USA, Prentice - Hall International, 1994.

Cox, B. J. **Object-Oriented Programming: An Evolutionary Approach.** Addison- Wesley, Massachusetts, 1987.

Çubukçu, Faruk. **Turbo Pascal Programlama Dili.** İstanbul, Türkmen Kitabevi, 1991.

Demiralp, Fehmi. **Turbo ve Borland C++ İle Nesneye Dayalı Programlama.** İstanbul, Beta Basım Yayım, 1993.

Demurlian, S, A. Beshers, G, M. ve Ting,T,C. "Programming Versus Databases in The Object-Oriented Paradigm". **Information and Software Technology.** No:2. USA, Butterworth Ltd, 1993.

Dittrich, K. R. **Object-Oriented Database System: The notions and the Issues.** New York, Computer Science Press, 1986.

Doğaç, A. Özsu, M, T. Bılırıs, A. ve Sellis, T. **Object Oriented Database Systems.** Kuşadası, Tubitak, 1993.

Erdun, Hakan. **Turbo ve Borland C & Pascal ile Grafik.** İstanbul, Beta Basım Yayım, 1993.

Esen, H, Öner. **İşletme Yönetiminde Sistem Yaklaşımı.** İstanbul, Bayrak Matbaacılık, 1985.

Fairley, R. "A Model of Software Structure". **Proc. 17th Hawaii Intl. Conf. on System Sciences.** Western Periodicals, 1984.

Fairley, R. **Software Engineering Concepts.** NewYork, McGraw-Hill, 1993.

Filman, R. ve Friedman, D. P. **Coordinated Computing: Tools and Techniques for Distributed Software.** New York, McGraw-Hill, 1984.

Florentin, J, J. **Object Oriented Programming Systems Tools and Applications.** Newyork, Chapman & Hall, 1991.

Freeman, P. "Fundamentals off Design". **IEEE Tutorial on Softwaare Design Techniques.** New York, Addison-Wesley, 1983.

Ghezzi, C. ve Jazayeri, M. **Programming Language Conceps.** NewYork, John Wiley & Sons, 1987.

Goldberg, A. **Smalltalk-80 The Interactive Programming Environment.** USA, Wesley Series in Computer Science, 1984.

Goldberg, A. ve Rabson, D. **Smalltalk-80 The Language.** USA, Wesley Series in Computer Science, 1989.

Gremillion, Lee. ve Philip, J. Pyburn. " Justifying Decision Support Systems and Office Administration Systems". **Journal of Management Information Systems.** New York, Addison-Wesley, 1985.

Gunther, R. **Management Methodology for Software Product Engineering.** New York, Wiley Interscience, 1978.

Güven, Doğan. **Bilişim Dizgilerinin Çözümleme Tasarım ve Gerçekleştirimi için Görsel Nesneye Dayalı bir CASE ortamı.** İzmir, Basılmamış Doktora Tezi, 1991.

Hakman, Sina. ve Üncü, A. Nilgün. "Nesneye Dayalı Sistemler ve Kullanıcı Programları". **Bildiriler.** İstanbul, Tayf Matbaacılık, 1992.

Halbert, D, C. ve O'Brien, P, D. "Using Types and Inheritance in Object-Oriented Languages". **European Conference on Object-Oriented Programming**. Paris, 1987.

Heintz, Timothy, J. "An Object-Oriented Approach to planning and Managing Software Development Projects". **Information & Management.** New York, McGraw-Hill, 1991.

Heitz, M. ve Labreville, B. "Designand Development of Distributed Software Using Hierarchical Object-Oriented Design and Ada". **Ada-Europe International Conference.** Munich. 1988.

Henry, C. Lucos. **Introduction to Computers and Informtion Systems.** New York, Macmillan Publishing Company, 1986.

Huang, J. A. ve Galiana, F.D. "A Integrated Personal Computer Graphics Environment for Power System Education, Analysis and Design". **IEEE Transactions on Power Systems.** New York, McGraw-Hill, 1991.

Hunt, V, Daniel. **Artificial Intelligence & Expert Systems Sourcebook.** Newyork, Chapman & Hall, 1986.

Jackson, M. A. **System Development.** Englewood Cliffs, Prentice-Hall, 1983.

Keyes, Jessica. "Languages: The New Generation". **Computerworld.** New York, McGraw-Hill, 1990.

Kut, R, Alp. **Bir Veri Tabanı Yönetim Sistemi İçin Geliştirilmiş Nesne Kökenli Kullanıcı Arayüzü.** İzmir, Basılmamış Doktora Tezi, 1991.

Li, S. ve Shahidehpour, S, M. "An Object Oriented Power System Graphics Package for Personal Computer Environment". **IEEE Transactions on Power Systems.** No:3. USA, IEEE, 1992.

Locus, H. C. **The Analysis, Design, and Implementation of Informations Systems.** New York, McGraw-Hill, 1992.

Margaret, M, B. ve Allen, L, A. "Declarative Visual Languages". **Journal Of Visual Languages and Computing.** No:5. USA, Academic Press Limited, 1994.

Martin, James. **Fourth-Generation Languages.** Englewood Cliffs, Prentice-Hall, 1985.

McCorduck, P. ve Feignebaum, E. **The Fifth Generation.** New York, Addison-Wesley, 1983.

Meyer, B. **Object-Oriented Software Construction.** U.S.A., Prentice-Hall, 1988.

Millman, Howard. "There's Something for All in Applications Generators". **Computerworld.** New York, McGraw-Hill, 1992.

Morris, Stephen. **Object Oriented Programming Under Windows.** British Library, Butterworth - Heinemann Ltd, 1994.

Paepcke, Andreas. **Object Oriented Programming the Clos Perspective.** USA, Teach Books, 1993.

Parker, S. ve Case, T. **Management Information Systems Strategy and Action.** NewYork, McGraw-Hill, 1993.

Peterson, J. L. ve Silbershatz, A. **Operating System Concepts.** New York, Addison-Wesley, 1985.

Pratt, T, W. **Programming Languages:Design and Implementation.** U.S.A., Prentice-Hall, 1984.

Rambaugh, J. Michael, B. William P. ve Frederick E. **Object-Oriented Modeling and Design.** Englewood Cliffs, Prentice-Hall, 1991.

Robinson, Peter. **Object Oriented Design**. Newyork, Chapman & Hall, 1992.

Rumbaugh, J. **Object-Oriented Modeling and Design.** U.S.A., Prentice-Hall, 1990.

Sarıdoğan, M, Erhan. **C++ ve Nesneye Yönelik Programlama.** İstanbul, Sistem Yayıncılık, 1994.

Savic, Düsko. **Object Oriented Programming With SmallTalk / V.** England, Ellis Horwood Limited, 1990.

Shaw, A. "Software Specification Languages Based on Regular Expressions". **in Software Development Tools.** Berlin, Springer-Verlag, 1980.

Shelly, G. Cashman, T. Waggoner, G. ve Waggoner, W. **Complete Computer Conceps and Microcomputer Applications.** U.S.A., Boyd & Fraser publishing company, 1992.

Smedema, C. H. **The Programming Languages Pascal,Modula, CHILL, and Ada.** Englewood Cliffs, Prentice-Hall, 1983.

Stein, L, A. Lieberman,H. ve Ungar,D. "A Shared View of Sharing: The Treaty of Orlando". **In:Object-Oriented Concepts, Databases and Applications.** New York, ACM Press, 1989.

Synder, A. "Common Objects:An Overview". **Prooceedings of The Object-Oriented Programming Workshop.** NewYork, ACM Press, 1986.

Tello, Ernest, R. **Object Oriented Programming for Artificial Intelligence.** Newyork, Wesley Publishing Company, 1989.

Tom, Gilb. **Principles of Software Engineering Management.** New York, Addison-Wesley, 1988.

Voss, Greg. **Object Oriented Programming: An Introduction.** Tokyo, Osborne McGraw - Hill, 1991.

Wachal, R. W. Tarnawecky, M. Z. ve Swift, G.W. "A Power System Analysis Package for Students Using Computer Graphics". **IEEE Transactions on Power Apparatus and Systems.** New York, McGraw-Hill, 1983.

Wegner, P. "Clasification in Object-Oriented Systems". **Proceeding of the Object-Oriented Workshop.** New York, Sigplan, 1986.

Yanık, Memik. **Borland Delphi ile Görsel Programcılık.** İstanbul, Sistem Yayıncılık, 1996.

Yoğurtçu, Tendü. "Nesneye Dayalı Çözümleme İçin Yeni Bir Gösterim Biçimi". **Bildiriler.** İstanbul, Tayf Matbaacılık, 1992.

Yu, D. C. Chen, S. T. ve Bischke, R. F. "A Pc Oriented Interactive And Graphical imulation Package for Power System Study". **IEEE Transactions on Power Systems.** New York, McGraw-Hill, 1989.

EKLER

Ek. 1. Cizim.Pas (Ana Program)

```
uses
  crt,dos,graph,drivers,turbo3,fkullan,tus,pencere1,fkonbel1;

const
  Fonts : array[0..10] of string[11] =
  (' Default ',' Triplex ',' Small   ',' SansSerif ',' Gothic ',
   ' Script  ',' Simplex ','Trip.Script',' Complex ',
   ' European ',' Bold  ');

type
   dugum=record
      ptr:pointer;
      boy:integer;
      x,y,ax,ay:integer;
     end;
  dizi=array[1..2] of dugum;

var
        tuslar    :array[1..20] of buton;
        x,y       :integer;
        i,j,sonuc,fnkdon:byte;
        s1,s2     :string;
        oldpalette :palettetype;
        gecerliharfbuyuk,sayi :byte;
        aktifyazitipino  :byte;
        acikdosyadegisti,kabul,advar,test:boolean;
        dosyaadi:string;

        {__Grafik sürücüsünün var olup olmadığını test eder}

        procedure grafik_kur;
                var gs,gm:integer;
                begin
                                gs:=detect;
                                gm:=0;
                        initgraph(gs,gm,'c:\bp\bgi');
                        if GraphResult <> grOk then begin
                        writeln('GRAFİK SÜRÜCÜNÜZ YOK.....');
                        delay(500);
                        halt(1);
                end;
        end;{grafik_kur}
        {__Fare sürücüsünün var olup olmadığını test eder}
```

```
        procedure faretest;
                var test:boolean;
                begin
                        test:=farevar;
                        if not test then begin
                writeln('FARE SÜRÜCÜNÜZ KURULMAMIŞ.....');
                delay(500);
                halt(1);
                end;
        end;{fare test}

procedure ilkekran;
var p1:pointer;
   s1:integer;
   f:file;
begin
 setbkcolor(0);
 clearviewport;
 {$I-}
 Assign (F,'ilkek.1MG');
 Reset (F,1);
 if (IOResult <> 0) then exit;
 {$I+}
 s1  :=FileSize (F);
 GetMem (p1,s1);
 Blockread (F,p1^,s1);
 Close (F);
 PutImage (57,33, p1^,CopyPut);
 FreeMem(p1,s1);

 {$I-}
 Assign (F,'ilkek.2MG');
 Reset (F,1);
 if (IOResult <> 0) then exit;{dosya yok}
 {$I+}

 s1  :=FileSize (F);
 GetMem (p1,s1);
 Blockread (F,p1^,s1);
 Close (F);
 PutImage (314,33, p1^,normalput);
 FreeMem(p1,s1);
{Kapak başı}
 setcolor(3);
 rectangle(0,0,getmaxx,getmaxy);
 setcolor(4);
 rectangle(56,33,571,440);
```

```
 settextstyle(1,0,2);
setcolor(4);
outtextxy(220,18,'PROGRAM ADI   ');
settextstyle(3,0,2);
outtextxy(150,433,'ATATURK UNIVERSITESI - ERZURUM');
settextstyle(3,0,2);
outtextxy(12,50,'Y');
outtextxy(12,120,'Z');
outtextxy(16,150,'I');
outtextxy(12,180,'Y');
outtextxy(12,210,'A');
outtextxy(12,280,'A');
outtextxy(12,310,'Y');
outtextxy(16,340,'I');
outtextxy(12,370,'K');

outtextxy(610,50,'1');
outtextxy(610,155,'9');
outtextxy(610,260,'9');
outtextxy(610,370,'6');

putpixel(202,436,3);
putpixel(203,436,3);
putpixel(211,436,3);
putpixel(210,436,3);

putpixel(248,436,3);
putpixel(249,436,3);
putpixel(256,436,3);
putpixel(257,436,3);

putpixel(272,436,3);
putpixel(273,436,3);

putpixel(324,436,3);
putpixel(325,436,3);

putpixel(364,436,3);
putpixel(365,436,3);

putpixel(16,151,3);
putpixel(16,152,3); {Kapak sonu}
settextstyle(0,0,0);

end;{ilkekran}

{**********************}
```

```
{__Ekran zemini ve ortadaki çalışma alanını oluşturur}
procedure ekran_zemini;
begin
 setbkcolor(0);
 cleardevice;
 setfillstyle(1,7);
 bar(0,0,getmaxx,getmaxy);

 setcolor(14);
 rectangle(56,32,571,440);
   {iç renk değişimi}
 setfillstyle(1,3);
 bar(57,33,570,439);
end; {ekran_zemini}

{__Sol taraftaki renk paletini çizer}
procedure renkler;
var i,j:byte;
   er:byte;
begin
  er:=getcolor;{varolan yazı rengini alırız}
  for i:=0 to 15 do begin
  setfillstyle(1,i);
  bar(3,3+i*22,48,24+i*22);
  end;
  aktifrenk(0);
  setcolor(15);
  outtextxy(6,360,'Aktif');
  setcolor(5);
  rectangle(0,0,51,356);
  rectangle(1,1,50,355);
  rectangle(2,2,49,354);
  setcolor(4); {aktifrenk çerçevesi}
  rectangle(0,371,53,395);
  setcolor(er);{eski yazı rengini tekrar geçerki yaparız }
end;
{__Ekranın ortasına bir pencere açar. Yazı degişkeniyle gönderilen
 stringi pencerenin ortasına yazar.Altına ise EVET ve HAYIR yazarak
 tercih yapılmasını bekler.Fonksiyon, fare ile evet yazılı alana tık-
 lanırsa geriye true, hayır yazılı alana tıklanırsa false döndürür.}

function onay(yazi:string):boolean;
var
sb,xf,yf :integer;
eski:textsettingstype;
er  :byte;
cik :boolean;
begin
 repeat until not ftusubasili;
```

```
{varolan ayarlamaları unutmamak için alıyoruz}
er:=getcolor;
gettextsettings(eski);
{ekranın ortasına pencere çizilir}
fare(gizle);

sb:=pencereac(yazi,170,200,430,260);

{ Degisiklik yazısında s,g nin  ş ve ğ yapılması}
 if yazi[1]='D' then
 begin
 setcolor(15);
 outtextxy(202,188,'.');
 outtextxy(204,188,'.');
 outtextxy(218,204,'.');
 outtextxy(218,206,'.');
 end;
 setfillstyle(1,7);
 setcolor(0);
 bar(225,235,280,257);
 bar(319,235,375,257);
 outtextxy(230,235,'EVET');
 outtextxy(320,235,'HAYIR');
 cik:=false;
 fare(goster);
 {fare ile tercih yapılması beklenir}
 repeat
  repeat until ftusubasili;
  repeat until not ftusubasili;
  xf:=farex; yf:=farey;
  if((yf>235) and (yf<257)) then
   if((xf>225) and (xf<280)) then begin onay:=true;cik:=true;end
   else if((xf>319) and (xf<375)) then begin onay:=false;cik:=true;end;
 until cik;

 repeat until not ftusubasili;
 {seçilen tercihin alanı beyazlatılır}
 fare(gizle);
 setfillstyle(1,15);
 setcolor(0);
 if((xf>225) and (xf<280)) then
  begin
    bar(225,235,280,257);
    outtextxy(230,235,'EVET');
  end
 else begin
  bar(319,235,375,257);
  outtextxy(320,235,'HAYIR');
 end;
```

```
 fare(goster);
 delay(10);
 {eski ayarlamalar tekrar geçerli kılınır}
 fare(gizle);
 pencerekapat(170,200,sb);
 fare(goster);
 settextstyle(eski.font,eski.direction,eski.charsize);
 setcolor(er); {er ... etkin renk  }
end;

{__Ekranın ortasına pencere açarak bir dosya adı girilmesini
 ister. En fazla 8 karakterlik bir dosya adı kabul edilir.
 Enter ile girilen ad onaylanır. Escape tuşuna basılırsa
 dosya adı okuma işlemi iptal edilir. Okunan ad, ad stringi
 ile geri döndürülür.}
procedure dosyaadi_oku(var ad:string;var harfsayisi:byte);
var eski:textsettingstype;
    er,harfno:byte;
    harf:char;
    bb:integer;
    cik:boolean;
    event2:tevent;
    kcod    :word;
    ccod    :char;
    scod    :byte;
begin
 cik:=false; harfno:=0;
 er:=getcolor;
 gettextsettings(eski);
 harfno:=0;
 bb:=pencereac('Dosya ismini giriniz',200,200,410,260);
 setfillstyle(1,15);
 bar(210,235,400,255);
 setcolor(0);
 {ad değişkenini sıfırla}
 delete(ad,1,length(ad));
 event2.what:=0;
 repeat
 while (event2.what and evkeydown=0) do getkeyevent(event2);
 begin
    scod:=event2.scancode;
    ccod:=event2.charcode;
    kcod:=event2.keycode;
    case scod of
     14 :begin
          if harfno>0 then harfno:=harfno-1;
          bar(215+harfno*14,235,228+harfno*14,255);
        end;{ soltab}
     28 :if harfno>0 then cik:=true;{enter}
```

```
    else if((scod in[2..12,16..27,30..38,44..50]) and
          (harfno < 8) and (not (ccod in['*','+','=','?','é','/']))) then
        begin
        ccod:=upcase(ccod);
        outtextxy(215+harfno*14,233,ccod);
        harfno:=harfno+1;
        {ad:=ad+ccod;}
        ad[harfno]:=ccod;
        end;
   end;{case}
  end;{while}
  event2.what:=0;
until ((event2.scancode=1) or cik);
ad[0]:= #12;
if not cik then ad[0]:=#9; {'vazgec';}
harfsayisi:=harfno;
pencerekapat(200,200,bb);
settextstyle(eski.font,eski.direction,eski.charsize);
setcolor(er);
end;{dosya adı oku}

{__Çalışma alanını iki parçaya bölerek dad ile verilen dosya
 adı ile 2 dosya halinde saklar}
procedure ekransakla(dad:string);
var p1,p2:pointer;
    s1,s2:integer;
    s:string;
    f:file;
begin
    fare(gizle);
    s1:=imagesize(57,33,313,439);
    getmem(p1,s1);
    getimage(57,33,313,439,p1^);

    s2:=imagesize(314,33,570,439);
    getmem(p2,s2);
    getimage(314,33,570,439,p2^);

    dad[sayi+1]:='.';  {boşlukları varsa onlardan sonraya ekliyor}
    dad[sayi+2]:='1';
    dad[sayi+3]:='m';
    dad[sayi+4]:='g';

  {  Assign (F,dad+'.1MG');}
    Assign (F,dad);
    Rewrite (F,1);
    BlockWrite (F,p1^,s1);
    FreeMem (p1,s1);
    close(f);
```

```
    else if((scod in[2..12,16..27,30..38,44..50]) and
          (harfno < 8) and (not (ccod in['*','+','=','?','é','/']))) then
        begin
        ccod:=upcase(ccod);
        outtextxy(215+harfno*14,233,ccod);
        harfno:=harfno+1;
        {ad:=ad+ccod;}
        ad[harfno]:=ccod;
        end;
   end;{case}
  end;{while}
  event2.what:=0;
until ((event2.scancode=1) or cik);
ad[0]:= #12;
if not cik then ad[0]:=#9; {'vazgec';}
harfsayisi:=harfno;
pencerekapat(200,200,bb);
settextstyle(eski.font,eski.direction,eski.charsize);
setcolor(er);
end;{dosya adı oku}

{__Çalışma alanını iki parçaya bölerek dad ile verilen dosya
 adı ile 2 dosya halinde saklar}
procedure ekransakla(dad:string);
var p1,p2:pointer;
   s1,s2:integer;
   s:string;
   f:file;
begin
   fare(gizle);
   s1:=imagesize(57,33,313,439);
   getmem(p1,s1);
   getimage(57,33,313,439,p1^);

   s2:=imagesize(314,33,570,439);
   getmem(p2,s2);
   getimage(314,33,570,439,p2^);

   dad[sayi+1]:='.';  {boşlukları varsa onlardan sonraya ekliyor}
   dad[sayi+2]:='1';
   dad[sayi+3]:='m';
   dad[sayi+4]:='g';

  {  Assign (F,dad+'.1MG');}
   Assign (F,dad);
   Rewrite (F,1);
   BlockWrite (F,p1^,s1);
   FreeMem (p1,s1);
   close(f);
```

```
   dad[sayi+2]:='2';
   {Assign (F,dad+'.2MG');}
   Assign (F,dad);
   Rewrite (F,1);
   BlockWrite (F,p2^,s2);
   FreeMem (p2,s2);
   close(f);
  fare(goster);
end;{ekransakla}
{__Dad ile verilen adla kaydedilmiş dosyayı getirir}

procedure ekrangetir(dad:string; var sonuc:boolean);
var p1:pointer;
   s1:integer;
   f:file;
begin
 sonuc:=true;
 fare(gizle);

 dad[sayi+1]:='.';
 dad[sayi+2]:='1';
 dad[sayi+3]:='m';
 dad[sayi+4]:='g';

 {$I-}
 Assign (F,dad);
 Reset (F,1);
 if (IOResult <> 0) then begin fare(goster);
                        sonuc:=false;
                        exit;
                       end;{dosya yok}
 {$I+}
 s1  :=FileSize (F);
 GetMem (p1,s1);
 Blockread (F,p1^,s1);
 Close (F);
 PutImage (57,33, p1^,CopyPut);
 FreeMem(p1,s1);

 {$I-}
 dad[sayi+2]:='2';
 Assign (F,dad);
 Reset (F,1);
 if (IOResult <> 0) then begin fare(goster);
                         sonuc:=false;
                         exit;
                        end;{dosya yok}
 {$I+}
 s1  :=FileSize (F);
```

```
 GetMem (p1,s1);
 Blockread (F,p1^,s1);
 Close (F);
 PutImage (314,33, p1^,CopyPut);
 FreeMem(p1,s1);
fare(goster);
end;{ekrangetir}


{*****************}
procedure yenisayfa;
var
cevap :boolean;
begin
if acikdosyadegisti then
 begin
   cevap:=onay('Degisiklik kaydedilsin mi?');
   if cevap then
   begin
     if not advar then begin
                    fare(gizle);
                    dosyaadi_oku(dosyaadi,sayi);
                    if dosyaadi[0]=#9{'vazgec'} then begin fare(goster);
                                                           exit;
                                                     end;
                    {fare(goster);}
                    end;
     ekransakla(dosyaadi);
   end;
     fare(goster);

 end;
    fare(gizle);
    setcolor(14);
    rectangle(56,32,571,440);
    setfillstyle(1,3);
    bar(57,33,570,439);
    fare(goster);
    acikdosyadegisti:=false;
    advar:=false;
end;
{*****************}
procedure dosyaac;
var
cevap :boolean;
begin
if acikdosyadegisti then
 begin
   cevap:=onay('Eskisi kaydedilsin mi?');
```

```
  if cevap then
  begin
   if not advar then begin
                    fare(gizle);
                    dosyaadi_oku(dosyaadi,sayi);
                    if dosyaadi=#9{'vazgec'} then begin fare(goster);
                                                    exit;
                                                   end;
                    fare(goster);
                    end;
    ekransakla(dosyaadi);
  end;
    fare(goster);

 end;
 {acılacak dosyanın adını oku}
 fare(gizle);
 dosyaadi_oku(dosyaadi,sayi);
 if dosyaadi=#9{'vazgec'} then begin fare(goster);exit;end;
 advar:=true;
 fare(goster);
 acikdosyadegisti:=false;

 ekrangetir(dosyaadi,cevap);
 if not cevap then begin
                  advar:=false;
                  end;

end;
{*****************}
procedure kaydet;
var
cevap :boolean;
begin
if acikdosyadegisti then
 begin
   cevap:=onay('Emin misiniz?');
   if cevap then
   begin
    if not advar then begin
                    fare(gizle);
                    dosyaadi_oku(dosyaadi,sayi);
                    if dosyaadi=#9{'vazgec'} then begin fare(goster);
                                                    exit;
                                                   end;
                    advar:=true;
                    fare(goster);
                   end;
    ekransakla(dosyaadi);
```

```
   fare(goster);
   acikdosyadegisti:=false;
  end;
 end;
end;{kaydet}
{*****************}
procedure yeniadver;
var
cevap :boolean;
begin
  if advar then
   begin
   fare(goster);
   cevap:=onay('Emin misiniz?');
   if cevap then
   begin
     fare(gizle);
     dosyaadi_oku(dosyaadi,sayi);
     fare(goster);
     if dosyaadi=#9{'vazgec'} then exit;
     ekransakla(dosyaadi);
     acikdosyadegisti:=false;
   end;
  advar:=true;
  end;
end;
{*****************}
procedure yazici;
var
cevap :boolean;
r:registers;
p1,p2,p3,p4:pointer;
s1,s2,s3,s4:word;
begin
   fare(goster);
   cevap:=onay('Emin misiniz?');
   if cevap then
   begin
    fare(gizle);
   {kenarları saklıyoruz}
    setfillstyle(1,15); {doldurma çeşidi,rengi}

    s1:=imagesize(0,0,570,32);
    getmem(p1,s1);
    getimage(0,0,570,32,p1^);
    bar(0,0,570,32);

    s2:=imagesize(0,33,56,getmaxy);
    getmem(p2,s2);
```

```
    getimage(0,33,56,getmaxy,p2^);
    bar(0,33,56,getmaxy);

    s3:=imagesize(57,440,getmaxx,getmaxy);
    getmem(p3,s3);
    getimage(57,440,getmaxx,getmaxy,p3^);
    bar(57,440,getmaxx,getmaxy);

    s4:=imagesize(571,0,getmaxx,439);
    getmem(p4,s4);
    getimage(571,0,getmaxx,439,p4^);
    bar(571,0,getmaxx,439);

   {yazdırma kesmesini çağırıyoruz}
    fillchar(r,sizeof(r),0);
    r.ah:=$5;
    intr($5,r);
    putimage(0,0,p1^,normalput);
    putimage(0,33,p2^,normalput);
    putimage(57,440,p3^,normalput);
    putimage(571,0,p4^,normalput);
    FreeMem(p1,s1);
    FreeMem(p2,s2);
    FreeMem(p3,s3);
    FreeMem(p4,s4);
    fare(goster);
   end;
end;{yazici}
{*****************}
procedure tus_koordinatlarini_ata;
var i,j:integer;
    s1:string;
begin
 for i:=1 to 8 do begin
 tuslar[i].baslat;
 tuslar[i].numara:=i;
 str(i,s1);
 tuslar[i].desendosyasi:='c:\bp\tuslar\tus'+s1+'.img';
 tuslar[i].butonx1:=getmaxx-60;
 tuslar[i].butony1:=4+i*59-59;
 tuslar[i].butonx2:=getmaxx-10;
 tuslar[i].butony2:=54+i*59-59
 end;
 {ust tusların yatay koordınatları atanıyor}
 for i:=9 to 13 do   begin
  tuslar[i].baslat;
  tuslar[i].numara:=i;
  str(i,s1);
  tuslar[i].desendosyasi:='c:\bp\tuslar\tus'+s1+'.img';
```

```
 tuslar[i].butonx1:=60+(i-9)*103;
 tuslar[i].butony1:=4;
 tuslar[i].butonx2:=156+(i-9)*103;
 tuslar[i].butony2:=25;
end;

tuslar[20].desendosyasi:='c:\bp\tuslar\tus20.img';
tuslar[20].butonx1:=4;
tuslar[20].butony1:=getmaxy-65;
tuslar[20].butonx2:=45;
tuslar[20].butony2:=467;
{alt tuslarım yatay koordınatları atanıyor}
for i:=14 to 19 do   begin
 tuslar[i].numara:=i;
 str(i,s1);
 tuslar[i].desendosyasi:='c:\bp\tuslar\tus'+s1+'.img';
 tuslar[i].butony1:=459;
 tuslar[i].butony2:=475;
end;

{ alt tuşların koordinatları}
i:=14;
while i<20 do begin
 j:=145+(i-14)*45;
 tuslar[i].butonx1:=j;
 tuslar[i].butonx2:=j+15;
 i:=i+2;
end;

i:=15;
while i<20 do begin
 if i=19 then j:=225+(i-14)*45
 else j:=155+(i-14)*45;
 tuslar[i].butonx1:=j;
 tuslar[i].butonx2:=j+15;
 i:=i+2;
end;
     { tusları ilk olarak çizer}
for i:=1 to 20 do begin
 tuslar[i].kalinlik:=2;
 tuslar[i].ciz;
 tuslar[i].donat;
 if i<9 then  tuslar[i].led(12,4,5,10,8,5);
end;
end;{tuş koordinatlarını ata}
{*****************}
procedure altpencereler;
var er:byte;
begin
```

```
 er:=getcolor;
 setcolor(15);
 outtextxy(143,445,'Çizgi Kal.');
 outtextxy(232,445,'Harf Büyük.');
 outtextxy(362,445,'Harf Tipi');
 setfillstyle(1,15);
 bar(164,458,196,476);
 bar(255,458,286,476);
 bar(344,458,445,476);
 setcolor(5);
 rectangle(141,456,219,478);
 rectangle(231,456,309,478);
 rectangle(321,456,469,478);
 setcolor(er);
end;{ altpencereler ve iç boyaması}

{*****************}
procedure aktifyazitipi(islem:byte);
begin
 if islem=1 then
  begin
   if aktifyazitipino=10 then aktifyazitipino:=0
   else inc(aktifyazitipino);
   settextstyle(aktifyazitipino,
              horizdir,gecerliharfbuyuk);
  end
 else
  begin
   if aktifyazitipino=0 then aktifyazitipino:=10
   else dec(aktifyazitipino);
   settextstyle(aktifyazitipino,
              horizdir,gecerliharfbuyuk);
   end;
end;{aktifyazitipi}


procedure yazitipiyaz;
var eski:textsettingstype;
    er:byte;
begin
er:=getcolor;
gettextsettings(eski);

setfillstyle(1,15);
bar(344,458,445,476);

setcolor(0);
settextstyle(0,0,1);
```

```
outtextxy(353,465,fonts[aktifyazitipino]);
settextstyle(eski.font,eski.direction,eski.charsize);
setcolor(er);
end;

procedure harfbuyukyaz;
var eski:textsettingstype;
   er:byte;
   s3:string;
begin
er:=getcolor;
gettextsettings(eski);
settextstyle(0,0,1);

setfillstyle(1,15);
bar(255,458,286,476);

setcolor(0);
str(gecerliharfbuyuk,s3);
outtextxy(267,465,s3);
settextstyle(eski.font,eski.direction,eski.charsize);
setcolor(er);

end;

procedure cizkalinyaz;
var eski:textsettingstype;
   er:byte;
   s3:string;
begin
er:=getcolor;
gettextsettings(eski);
settextstyle(0,0,1);

setfillstyle(1,15);
bar(164,458,196,476);
setcolor(0);
str(gecerlicizgik,s3);
outtextxy(177,465,s3);
settextstyle(eski.font,eski.direction,eski.charsize);
setcolor(er);
end;

{-*********************-}
procedure baskatus(tno:byte);
{-*********************-}
procedure kalem;
 var eskix1,eskiy1,x,y:integer;
    sonuc,t,ii,jj:byte;
```

```
begin
fare(goster);
repeat
 repeat until not ftusubasili;{girişte tuştan kaldırmasını beklemek}
 repeat
  repeat until ftusubasili;
  eskix1:=farex;eskiy1:=farey;
  farekonumubelirle(eskix1,eskiy1,sonuc,t);
  if (sonuc=5) and (t<>3) then exit;
  if sonuc in [4,6,7] then baskatus(t);
  delay(150);
 until sonuc=2;
 dikeysinir(33,439-gecerlicizgik);
 yataysinir(57,570-gecerlicizgik);
 setcolor(gecerlirenk);

 repeat
 x:=farex;y:=farey;
 fare(gizle);
 acikdosyadegisti:=true;
 for ii:=0 to gecerlicizgik do
  for jj:=0 to gecerlicizgik do
    line(eskix1+ii,eskiy1+jj,x+ii,y+jj);
 fare(goster);
 eskix1:=x;eskiy1:=y;
 until not ftusubasili;
 dikeysinir(0,getmaxy);
 yataysinir(0,getmaxx);

until false;
end;{kalem}

{**********************}
 procedure sakla1(x1,y1,x2,y2:integer; var d1:dizi);
 var l,boyx,i,gecici:word;
 begin
   if x2<x1 then begin  gecici:=x1; x1:=x2; x2:=gecici; end;
   if y2<y1 then begin  gecici:=y1; y1:=y2; y2:=gecici; end;

    boyx:=abs(x2-x1);
    l:=boyx div 2;

    d1[1].x:=x1;
    d1[1].y:=y1;
    d1[1].ax:=x2;
    d1[1].ay:=y2;
                          {}
    d1[1].boy:=imagesize(x1,y1,x1+l,y2);
    getmem(d1[1].ptr,d1[1].boy);
```

```
    getimage(x1,y1,x1+l,y2,d1[1].ptr^);
                  {}
    d1[2].x:=x1+l+1;
    d1[2].y:=y1;
    d1[2].boy:=imagesize(x1+l+1,y1,x2,y2);
    getmem(d1[2].ptr,d1[2].boy);
    getimage(x1+l+1,y1,x2,y2,d1[2].ptr^);

 end;{sakla1}
 {*****************}
 procedure sakla(x1,y1,x2,y2:integer; var d1:dizi);
 var l,boyx,i,gecici:word;
 begin
   if x2<x1 then begin  gecici:=x1; x1:=x2; x2:=gecici; end;
   if y2<y1 then begin  gecici:=y1; y1:=y2; y2:=gecici; end;

    boyx:=abs(x2-x1);
    l:=boyx div 2;

    d1[1].x:=x1;    d1[1].y:=y1;    d1[1].ax:=x2;    d1[1].ay:=y2;

d1[1].boy:=imagesize(x1,y1,x1+l+gecerlicizgik,y2+gecerlicizgik);
getmem(d1[1].ptr,d1[1].boy);
    getimage(x1,y1,x1+l+gecerlicizgik,y2+gecerlicizgik,d1[1].ptr^);

    d1[2].x:=x1+l+1;
    d1[2].y:=y1;
    d1[2].boy:=imagesize(x1+l+1,y1,x2+gecerlicizgik,y2+gecerlicizgik);
    getmem(d1[2].ptr,d1[2].boy);
    getimage(x1+l+1,y1,x2+gecerlicizgik,y2+gecerlicizgik,d1[2].ptr^);

 end;{sakla}
 {*****************}
 procedure olustur(var d2:dizi);
 var i:byte;
 begin
   for i:=1 to 2 do begin
    putimage(d2[i].x,d2[i].y,d2[i].ptr^,normalput);
    FreeMem (d2[i].ptr,d2[i].boy);
   end;
 end; {olustur}

{*************************************************}
procedure cizgi;
Var
  ptrdizi          :dizi;
  sayi,sonuc7,t7    :byte;
  i,x,y,x1,y1,ii,jj :integer;
begin
```

```
 fare(goster);
 repeat
   repeat until not ftusubasili;
   repeat
    repeat until ftusubasili;
    x:= farex; y:=farey;
    farekonumubelirle(x,y,sonuc7,t7);
    if sonuc7=5 then if t7<>7 then exit;
    if sonuc7 in [4,6,7] then baskatus(t7);
    delay(150);
   until sonuc7=2;
    yataysinir(57,570-gecerlicizgik);
    dikeysinir(33,439-gecerlicizgik);
    setcolor(gecerlirenk);

    repeat until ftusubasili;
    x:=farex; y:=farey;

    while ftusubasili do begin
     x1:=farex; y1:=farey;
     delay(30);
     fare(gizle);
     acikdosyadegisti:=true;
     sakla(x,y,x1,y1,ptrdizi);
     for ii:=0 to gecerlicizgik do
     for jj:=0 to gecerlicizgik do
       line(x+ii,y+jj,x1+ii,y1+jj);
     delay(30);{yokedici}
     olustur(ptrdizi);
     fare(goster);
    end;{while}
    fare(gizle);
    for ii:=0 to gecerlicizgik do
    for jj:=0 to gecerlicizgik do
      line(x+ii,y+jj,x1+ii,y1+jj);
    fare(goster);
    yataysinir(0,getmaxx);
    dikeysinir(0,getmaxy);

 until false;
 end;{cizgi}
{*************************************************}
{*************************}
procedure yay;
var x,y,x1,y1,xr,yr:integer;
    sonuc6,t6,s:byte;
    tdizi:dizi;
   secenek:byte;
begin
```

```
fare(goster);
repeat
 secenek:=1;
 repeat until not ftusubasili;
 repeat
  repeat until ftusubasili;
  x:=farex; y:=farey;
  farekonumubelirle(x,y,sonuc6,t6);
  if sonuc6=5 then if t6 <> 6 then exit;
  if sonuc6 in [4,6,7] then baskatus(t6);
  delay(150);
 until sonuc6=2;

 yataysinir(57,570-gecerlicizgik);
 dikeysinir(33,439-gecerlicizgik);
 setcolor(gecerlirenk);
 repeat
    x1:=farex;y1:=farey;
    xr:=(((x1-x) div 2)); yr:=(((y1-y) div 2 ));
    setcolor(gecerlirenk);
    fare(gizle);
    sakla(x-1,y-1,x1,y1,tdizi);
    fare(goster);
   acikdosyadegisti:=true;
    for s:=1 to gecerlicizgik do
  ellipse(x+trunc(xr),y+trunc(yr),0,360,trunc(abs(xr))-s,trunc(abs(yr))-s);
    if sagd then begin
  setfillstyle(1,gecerlirenk);
  fillellipse(x+trunc(xr),y+trunc(yr),trunc(abs(xr))-s,trunc(abs(yr))-s);
     secenek:=2;
    end;
    delay(50);
    fare(gizle);
    olustur(tdizi);
    fare(goster);
 until not ftusubasili;
 fare(gizle);
 for s:=1 to gecerlicizgik do
 ellipse(x+trunc(xr),y+trunc(yr),0,360,trunc(abs(xr))-s,trunc(abs(yr))-s);
 if secenek=2 then
 fillellipse(x+trunc(xr),y+trunc(yr),trunc(abs(xr))-s,trunc(abs(yr))-s);
 fare(goster);
 yataysinir(0,getmaxx);
 dikeysinir(0,getmaxy);

 until false;
end;
{*****************************}
procedure dortgen;
```

```
var boy2,x,y,x1,y1,gecici:integer;
    sonuc5,t5,s:byte;
    tdizi:dizi;
    p2:pointer;
    secenek:byte;
begin
 fare(goster);
 repeat
  secenek:=1;
  repeat until not ftusubasili;
  repeat
   repeat until ftusubasili;
   x:=farex; y:=farey;
   farekonumubelirle(x,y,sonuc5,t5);
   if sonuc5=5 then if t5 <> 5 then exit;
   if sonuc5 in [4,6,7] then baskatus(t5);
   delay(150);
  until sonuc5=2;
  yataysinir(57,570-gecerlicizgik);
  dikeysinir(33,439-gecerlicizgik);
  setcolor(gecerlirenk);

  while ftusubasili do begin
    x1:=farex;y1:=farey;
    fare(gizle);
    sakla(x,y,x1,y1,tdizi);

    if (x1<>x) and(y<>y1) then
    begin
     for s:=0 to gecerlicizgik do
      rectangle(tdizi[1].x+s,tdizi[1].y+s,
              tdizi[1].ax-s+gecerlicizgik,tdizi[1].ay-s+gecerlicizgik);
     setfillstyle(1,gecerlirenk);
     if(sagd)then begin bar(x,y,x1,y1); secenek:=2; end;
    end;
    fare(goster);
    delay(60);
    fare(gizle);
    olustur(tdizi);
    fare(goster);
  end; {while}
  fare(gizle);
  acikdosyadegisti:=true;
  for s:=0 to gecerlicizgik do
     rectangle(tdizi[1].x+s,tdizi[1].y+s,
             tdizi[1].ax-s+gecerlicizgik,tdizi[1].ay-s+gecerlicizgik);
  if secenek=2 then bar(x,y,x1,y1);
  fare(goster);
  yataysinir(0,getmaxx);
```

```
 dikeysinir(0,getmaxy);

 until false;
end;
{**********************************}
procedure zikzak;
var x,y,x1,y1,eskix,eskiy:integer;
    sonuc4,t4,test,ii,jj:byte;
    dizi4:dizi;
begin
  test:=0;
  fare(goster);
 repeat
  repeat until not ftusubasili;
   repeat
    repeat until ftusubasili;
    if test<>0 then begin x:=eskix; y:=eskiy;end
    else begin x:=farex;y:=farey;end;
    x1:=farex;y1:=farey;
    farekonumubelirle(x1,y1,sonuc4,t4);
    if sonuc4=5 then if t4<>4 then exit;
    if sonuc4 in [4,6,7] then baskatus(t4);
    delay(150);
   until sonuc4=2;
   test:=1;
   yataysinir(57,570-gecerlicizgik);
   dikeysinir(33,439-gecerlicizgik);
   setcolor(gecerlirenk);

   while ftusubasili do begin
    x1:=farex; y1:=farey;
    delay(50);
    fare(gizle);
    acikdosyadegisti:=true;
    sakla(x,y,x1,y1,dizi4);
    for ii:=0 to gecerlicizgik do
     for jj:=0 to gecerlicizgik do
       line(x+ii,y+jj,x1+ii,y1+jj);
    fare(goster);
    delay(50);
    fare(gizle);
    olustur(dizi4);
    fare(goster);
   end;
     eskix:=x1;eskiy:=y1;
     fare(gizle);
     for ii:=0 to gecerlicizgik do
      for jj:=0 to gecerlicizgik do
       line(x+ii,y+jj,x1+ii,y1+jj);
```

```
    fare(goster);

  yataysinir(0,getmaxx);
  dikeysinir(0,getmaxy);

 until false;
end;
{**********************************}
procedure kestasi;
var  x,y,x1,y1,ex,ey,eex,eey,
     dx,dy,m,l,d,s,boy1,boyx,l1:word;
     sonuc2,t2,ii,jj,sonuc21,t21:byte;
     dizi4,dizi5:dizi;
     ppp:pointer;
begin
 fare(goster);
 repeat
   repeat until not ftusubasili; {fare tusunun bırakılması beklenir}
   repeat
    repeat until ftusubasili; {fare tusuna basılması beklenir}
    x1:=farex;y1:=farey;
    farekonumubelirle(x1,y1,sonuc2,t2);
    if sonuc2=5 then if t2<>2 then exit;{cıkmadan dortgeni sil}
    if sonuc2 in [4,6,7] then baskatus(t2);
    delay(150);
   until sonuc2=2; {çalışma bölgesine tıklanmışsa devam}

   yataysinir(58,569);
   dikeysinir(34,438);
   setcolor(0);
  while ftusubasili do begin {fareye basılı olduğu sürece kesiksiz 4_gen çiz}
    ex:=farex; ey:=farey;
    fare(gizle);
    sakla1(x1,y1,ex,ey,dizi4);
    rectangle(x1,y1,ex,ey);
    fare(goster);
    delay(100);
    fare(gizle);
    olustur(dizi4);
    fare(goster);
  end;

   setlinestyle(1,0,0);  { kesikli dikdortgen cizmek icin}
   fare(gizle);
   sakla1(x1,y1,ex,ey,dizi4);
   rectangle(x1,y1,ex,ey);
   fare(goster);
   setlinestyle(0,0,0);
```

```
yataysinir(0,getmaxx);
dikeysinir(0,getmaxy);
{ }
repeat
 repeat until ftusubasili;
 eex:=farex;eey:=farey;
 farekonumubelirle(eex,eey,sonuc21,t21);
 if sonuc21=5 then if t21<>2 then {diger bir fonksiyon seçilmiş çık}
                      begin
                        olustur(dizi4);
                        exit;
                      end;
  if sonuc21 in [4,6,7] then begin repeat until not ftusubasili;
                           delay(10);baskatus(t21);end;
until (sonuc21=2) and (eex > dizi4[1].x) and (eex < dizi4[1].ax)
              and(eey>dizi4[1].y) and(eey<dizi4[1].ay);

m:=eex-dizi4[1].x;
l:=eey-dizi4[1].y;
s:=dizi4[1].ax-eex;
d:=dizi4[1].ay-eey;     { kesilen parcanın buyuklugune gore fare sınırla}
fare(gizle);
setfillstyle(1,3);
bar(x1,y1,ex,ey);
fare(goster);

yataysinir(58+m,569-s);
dikeysinir(34+l,438-d);
{hala üstteki repeatteki ftusu basili varsayılıyor}
while ftusubasili do begin  {sürükleme}
dx:=farex; dy:=farey;
fare(gizle);
sakla1(dx-m,dy-l,dx+s,dy+d,dizi5);

dizi4[1].x:=dx-m; dizi4[1].y:=dy-l;
dizi4[1].ax:=dx+s; dizi4[1].ay:=dy+d;
boyx:=abs(dizi4[1].ax-dizi4[1].x);
l1:=boyx div 2;
dizi4[2].x:=dizi4[1].x+l1+1;
dizi4[2].y:=dizi4[1].y;
olustur(dizi4);

setlinestyle(1,0,0);
rectangle(dx-m,dy-l,dx+s,dy+d);
setlinestyle(0,0,0);
fare(goster);
delay(30);
fare(gizle);
sakla1(dx-m,dy-l,dx+s,dy+d,dizi4);
```

```
  olustur(dizi5);
  fare(goster);
 end;{while}
  fare(gizle);
  acikdosyadegisti:=true;
  olustur(dizi4);
  setcolor(3);
  rectangle(dx-m,dy-l,dx+s,dy+d);
  fare(goster);
  yataysinir(0,getmaxx);
  dikeysinir(0,getmaxy);
 until false;
end;
{**************}
 procedure silgi;
  var x,y,eskix,eskiy:integer;
     k,sonuc1,t1:byte;
  begin
  fare(goster);
  repeat
   repeat until not ftusubasili;{girişte tuştan kaldırmasını beklemek}
   repeat
    k:=gecerlicizgik*2;
    repeat until ftusubasili;
    eskix:=farex;eskiy:=farey;
    farekonumubelirle(eskix,eskiy,sonuc1,t1);
    if (sonuc1=5) and (t1<>1) then exit;
    if sonuc1 in [4,6,7] then baskatus(t1);
    delay(150);
   until (sonuc1=2) and(eskix > 57+k)and(eskiy > 33+k)
                and(eskix < 570-k)and(eskiy < 439-k);
   k:=gecerlicizgik*2;
   dikeysinir(33+k,439-k);
   yataysinir(57+k,570-k);
   setfillstyle(1,3);
   setcolor(0);
   fare(gizle);

   while ftusubasili do begin
   x:=farex;y:=farey;
   rectangle(x-k,y-k,x+k,y+k);
   delay(5);
   bar(eskix-k,eskiy-k,eskix+k,eskiy+k);
   eskix:=x;eskiy:=y;
   end;

   fare(goster);
   acikdosyadegisti:=true;
   bar(eskix-k,eskiy-k,eskix+k,eskiy+k);
```

```
  dikeysinir(0,getmaxy);
  yataysinir(0,getmaxx);
 until false;
 end;{silgi}
{******************}
procedure yazi;
var
 x,y,posx,posy,
 sonx,sony,kurx,kury
            :integer;
 en,boy,sonuc8,t8:byte;
 event1       :Tevent;
 kcode        :word;
 ccode        :char;
 scode        :byte;

begin
 repeat until not ftusubasili;
  fare(goster);
  initevents;
  setcolor(gecerlirenk);
  settextstyle(aktifyazitipino,0,gecerliharfbuyuk);
 repeat

  while((event1.what and evkeydown=0)
  and not ftusubasili) do getkeyevent(event1);
  if ftusubasili then
  begin
    repeat until not ftusubasili;
    x:=farex; y:=farey;
    farekonumubelirle(x,y,sonuc8,t8);
    if sonuc8=5 then if t8 <> 8 then exit;
    if sonuc8 in [4,6,7] then baskatus(t8);
    setcolor(gecerlirenk);
    settextstyle(aktifyazitipino,0,gecerliharfbuyuk);
    en:=textwidth('m'); boy:=textheight('m');
    if sonuc8=2 then begin
                     posx:=x;posy:=y;
                     if posy<=(32+boy) then posy:=32+boy;
                     if posx<=56 then posx:=58;
                     if posy+boy >=439 then posy:=439-boy;
                     if posx+en >=569 then posx:=569-en;
                    end;
                   x:=posx; y:=posy; sonx:=x;sony:=y;
                   kurx:=x;kury:=y;
                   fare(gizle);
                   line(posx,posy+3,posx,posy+5);
                   fare(goster);
    delay(150);
```

```
end {if then}
else begin
  scode:=event1.scancode;
  ccode:=event1.charcode;
  kcode:=event1.keycode;

  case kcode of
   3592 :begin
         if posx >=58+en then posx:=posx-en;
         fare(gizle);
         setfillstyle(1,3);
         bar(posx,posy-(boy div 2),posx+en,posy+(boy div 2));
         fare(goster);
        end; { soltab}
   7181 :begin if posy <= (439-2*boy) then begin
                                    posx:=58;posy:=posy+boy;
                                   end;
             end;{enter}
   14624:begin if (posx < 570-2*en) then posx:=posx+en; end;{space}
   18176:begin posx:=x;posy:=y; end; {home}
   18432:begin if posy >= 33+boy then posy:=posy-boy;end;{yukarı ok}
   19200:begin if posx >= 58+en then posx:=posx-en;end;{sol ok}
   19712:begin if posx <= 569-2*en then posx:=posx+en;end;{sağ ok}
   20224:begin posx:=sonx; posy:=sony; end;{end}
   20480:begin if posy <= 439-2*boy then posy:=posy+boy;end;{aşağı ok}
   21248:begin
         fare(gizle);
         setfillstyle(1,3);
         bar(posx,posy-(boy div 2),posx+en,posy+(boy div 2));
         fare(goster);
        end;{delete}
   else if(scode in[2..13,16..27,30..41,43..53,86]) then
       begin
          fare(gizle);
          acikdosyadegisti:=true;
          outtextxy(posx,posy-(boy div 2),ccode);
          inc(posx,en);
          if posx >= 570-en then begin
                                 dec(posx,en);
                                 posx:=58;
                                 posy:=posy+boy;
                                 if posy >= 439-boy then dec(posy,boy);
                                end;
          sonx:=posx; sony:=posy;
          fare(goster);
       end;
    end;{case}
  end;{if else}
  setcolor(3);
```

```
  line(kurx,kury+3,kurx,kury+5);
  kurx:=posx;kury:=posy;
  setcolor(gecerlirenk);
  line(posx,posy+3,posx,posy+5);
  {kors. çiz}
  event1.what:=0;
 until event1.scancode=1;

end; {yazı}


{!******************!}

begin  {baskatus procedurunun govdesi}
 fare(gizle);
 tuslar[tno].resimliciz(1);
 fare(goster);
  case tno of
   1: silgi;
   2: kestasi;
   3: kalem;
   4: zikzak;
   5: dortgen;
   6: yay;
   7: cizgi;
   8: yazi;
   9 :yenisayfa;
   10:dosyaac;
   11:kaydet;
   12:yeniadver;
   13:yazici;
   14:begin  dec(gecerlicizgik);
             if gecerlicizgik=0 then gecerlicizgik:=12;
             cizkalinyaz;
        end;{cizgi kalınlığını azalt}
   15:begin   inc(gecerlicizgik);
             if gecerlicizgik=13 then gecerlicizgik:=1;
             cizkalinyaz;
        end;
   16:begin
             if gecerliharfbuyuk=0 then gecerliharfbuyuk:=8;
             dec(gecerliharfbuyuk);
             harfbuyukyaz;
        end; {harf büyük az}
   17:begin   inc(gecerliharfbuyuk);
             if gecerliharfbuyuk=8 then gecerliharfbuyuk:=0;
             harfbuyukyaz;
        end;
   18:begin
```

```
                aktifyazitipi(1);
                yazitipiyaz;
            end; {yazı stılı prev.}
    19:begin
                aktifyazitipi(0);
                yazitipiyaz;
            end;
    20:begin fare(goster); yenisayfa; closegraph;halt;end;
   end;{case}
  fare(gizle);
  tuslar[tno].resimliciz(0);
  fare(goster);
end;{ayar tusları}

{*****************************}
{************ ana program **************}
 begin
  grafik_kur;
  faretest;
  ilkekran;
  readln;
  ekran_zemini;
  renkler;
  tus_koordinatlarini_ata;
  altpencereler;
  acikdosyadegisti:=false;
  advar:=false;
  gecerlicizgik:=1;
  aktifyazitipino:=0;
  gecerliharfbuyuk:=1;
  gecerlirenk:=0;
  yazitipiyaz;
  cizkalinyaz;
  harfbuyukyaz;
  fare(goster);
  repeat
   repeat until ftusubasili;
   x:=farex; y:=farey;
   farekonumubelirle(x,y,sonuc,i);
   fare(gizle);
   if i in [1..20] then baskatus(i);
   fare(goster);
   delay(200);
  until false;

  closegraph;
 end.{ana program sonu}
```

Ek. 2. Tus.Pas (Unit)

```
unit tus;
interface
type  buton=object
        numara,kalinlik,durum:byte;  {durum butonun basılı(1) veya normal(0)
olup olmadığı}
        butonx1,butony1,
        butonx2,butony2:integer;
        desendosyasi   :string;
        procedure baslat;
        procedure ciz;
        procedure donat;
        procedure led(acikrenk,kapalirenk:byte;bx,by,ux,uy:integer);
        procedure resimliciz(onoff:byte);
        end;

implementation
 uses crt,graph;

 procedure buton.baslat;
 {var ss:string;}
 begin
  durum:=0;
 { str(numara,ss);}{bunu sil}
 end;

 procedure buton.donat;
 var
   size:word;
   ptr:pointer;
   f:file;
 begin
  {$I-}
  Assign (F,desendosyasi);
  Reset (F,1);
  if (IOResult = 0) then
  begin
   Size  :=FileSize (F);
   GetMem (ptr,Size);
   Blockread (F,ptr^,Size);
   Close (F);
   PutImage (butonx1+3,butony1+3, ptr^,3);
   FreeMem(ptr,Size);
  end;
 {$I+}
 end;

 procedure  buton.ciz;
```

```
var i,renk:integer;
   yenik:byte;
begin
 if durum=1 then yenik:=0
 else yenik:=kalinlik;
 renk:=getcolor;{aktif renk ayarını saklar}

 setcolor(lightgray); {ligtgray yerine numarasıda yazılabilir..}

 for i:=0 to yenik do
 begin
  setcolor(15);
  if false then setcolor(lightgray);
  line(butonx1+i,butony1+i,butonx2-i,butony1+i);
  line(butonx1+i,butony1+i,butonx1+i,butony2-i);
 end;

 setcolor(darkgray);
 for i:=0 to yenik do
 begin
  setcolor(8);
  if false then  setcolor(lightgray);
  line(butonx2-i,butony2-i,butonx2-i,butony1+i);
  line(butonx1+i,butony2-i,butonx2-i,butony2-i);
 end;

 setcolor(7);
 for i:=0 to butony2-butony1-2*yenik  do
 line(butonx1+yenik,butony1+yenik+i,
     butonx2-yenik,butony1+yenik+i);

 setcolor(5);
 for i:= 1 to 3 do rectangle(butonx1-i,butony1-i,butonx2+i,butony2+i);
 setcolor(0);
 rectangle(butonx1-1,butony1-1,butonx2+1,butony2+1);

 setcolor(15);
 if yenik <> 0 then
  line(butonx1,butony1,butonx1+yenik-1,butony1+yenik-1);

 setcolor(darkgray);
 line(butonx2,butony2,butonx2-yenik+1,butony2-yenik+1);

 setcolor(renk);{bu procedur. gelmeden önceki renk ayarını tekrar geçerli yap}
 end; { ciz }

procedure buton.led(acikrenk,kapalirenk:byte;bx,by,ux,uy:integer);
var  w:word;
begin
```

```
  w:=getcolor;
  case durum of
  0:setfillstyle(1,kapalirenk);
  1:setfillstyle(1,acikrenk);
  end;
  bar(butonx1+bx,butony1+by,butonx1+bx+ux,butony1+uy+by);
  setcolor(w);
 end;
 {*******************}
 procedure buton.resimliciz(onoff:byte);
 begin
  durum:=onoff;
  if onoff=1 then kalinlik:=0
  else kalinlik:=2;
  ciz;
  donat;
  if numara<9 then led(12,4,5,10,8,5);
 end;
end.
```

Ek. 3. Fkullan.Pas (Unit)

```
unit fkullan;
interface
 uses dos;
 const
   acik=true;
   kapali=false;
 type
   farefonk=(goster,gizle,sifirla);
 function Farevar:boolean;
 procedure fare(ffonk:farefonk);
 function farex:word;
 function farey:word;
 function sold:boolean;
 function sagd:boolean;
 procedure yataysinir(minx,maxx:word);
 procedure dikeysinir(miny,maxy:word);
 function ftusubasili:boolean;

implementation

 function farevar:boolean;
 var sseg,sofs:word;
 begin
  farevar:=true;
  sseg:=256*mem[0:51*4+3]+mem[0:51*4+2];
  sofs:=256*mem[0:51*4+1]+mem[0:51*4]+2;
  if mem[sseg:sofs-2]=$cf then  farevar:=false;
  if ((sseg=0) and (sofs=2)) then farevar:=false;
 end;

 procedure fare(ffonk:farefonk);
 begin
  if ffonk=sifirla then
   asm
    mov ax,0
    int 33h
   end;
  if ffonk=goster then
   asm
    mov ax,1
    int 33h
   end;
  if ffonk=gizle then
   asm
    mov ax,2
    int 33h
   end;
```

```
end;

procedure yataysinir(minx,maxx:word);
var r:registers;
begin
 r.ax:=7;
 r.cx:=minx;
 r.dx:=maxx;
 intr($33,r);
end;


procedure dikeysinir(miny,maxy:word);
var r:registers;
begin
 r.ax:=8;
 r.cx:=miny;
 r.dx:=maxy;
 intr($33,r);
end;

function farex:word;
var r:registers;
begin
 r.ax:=3;
 intr($33,r);
 farex:=r.cx;
end;


function farey:word;
var r:registers;
begin
 r.ax:=3;
 intr($33,r);
 farey:=r.dx;
end;


function sold:boolean;
var r:registers;
begin
 r.ax:=3;
 intr($33,r);
 if(r.bx and 1)= 1 then sold:=true
 else sold:=false;
end;

function sagd:boolean;
```

```
var r:registers;
begin
 r.ax:=3;
 intr($33,r);
 if(r.bx and 2)= 2 then sagd:=true
 else sagd:=false;
end;

 function ftusubasili:boolean;
begin
 if((sold) or (sagd)) then ftusubasili:=true
 else ftusubasili:=false;
end;
end.
```

Ek. 4. Fkonbel1.Pas (Unit)

```
unit fkonbel1;
interface  {----------}
 uses fkullan,crt,graph;
 var gecerlirenk,gecerlicizgik:byte;
  procedure farekonumubelirle(x,y:integer;var durum:byte;var tusno:byte);
  procedure aktifrenk(yenirenk:word);
implementation
 {*****************}
  procedure aktifrenk(yenirenk:word);
  var esk:fillsettingstype;
  begin
   getfillsettings(esk);
   setfillstyle(1,yenirenk);
   gecerlirenk:=yenirenk;
   bar(3,373,50,392);
   setfillstyle(esk.pattern,esk.color);
 end;{aktifrenk}

 {*******************}
 procedure farekonumubelirle(x,y:integer;var durum:byte;var tusno:byte);
 begin
  tusno:=50;
  durum:=0;{tusabasıldı or bilgi degisti}

  if ((x>4) and (x<47) and (y>5) and (y<352)) then
              begin durum:=3;aktifrenk(y div 22);end;

  if ((x>6) and (x<43) and (y>416)and(y<465)) then
                begin tusno:=20; durum:=4;end;


   if (x > getmaxx-60) and (x < getmaxx-10) then
   begin
    case (y-3) of
    2..51:tusno:=1;
    60..110:tusno:=2;
    119..169:tusno:=3;
    178..228:tusno:=4;
    237..287:tusno:=5;
    296..346:tusno:=6;
    355..405:tusno:=7;
    414..464:tusno:=8;
    end;
    if tusno in [1..8] then durum:=5;
   end;

   if (y > 461) and (y < 473) then
```

```
 begin
  case (x) of
  147..158:tusno:=14;
  202..213:tusno:=15;
  237..248:tusno:=16;
  292..303:tusno:=17;
  327..338:tusno:=18;
  452..463:tusno:=19;
  end;
  if tusno in [14..19] then durum:=6;
 end;

 if ((y > 6) and (y < 23)) then
 begin
  case (x) of
   62..154:tusno:=9;
  165..257:tusno:=10;
  268..360:tusno:=11;
  371..463:tusno:=12;
  474..566:tusno:=13;
  end;
 if tusno in [9..13] then durum:=7;
 end;

 if  (tusno=50) and (durum<>3) then durum:=1;{hıcbır tusa basılmadı}

 if ((x>56) and (x<571-gecerlicizgik)) then
  if((y>32)and(y<440-gecerlicizgik))then durum:=2;

{durum=0 enust,ayar ve cizim tusları hariç tusabasıldı ve tuskodu tusno`da,
 durum=1 tusabasılm2adı ve çalışma alanı da degil tuskodu hala 50 dır,
 durum=2 çalışma bölgesinde bir noktaya basıldı
 durum=3 renk degistirme ye basıldı
 durum=4 cıkıs tusuna basıldı
 durum=5 cizim tuslarından birine basıldı
 durum=6 ayar tusuna basıldı
 durum=7 en ust tuslara basıldı}

end;{farekonumubelirle}
end.
```

Ek. 5. Pencere1.Pas (Unit)

```
unit pencere1;
interface
 uses crt,dos,graph,fkullan;
 var
 bellek:integer;
 p:pointer;

 function pencereac(baslik:string;x1,y1,x2,y2:integer):integer;
 procedure pencerekapat(x1,y1,boy:integer);

implementation
 function pencereac(baslik:string;x1,y1,x2,y2:integer):integer;
 var  ks,i,j:integer;
 begin
  fare(gizle);
  pencereac:=imagesize(x1-9,y1-9,x2+9,y2+9);
  getmem(p,imagesize(x1-9,y1-9,x2+9,y2+9));
  getimage(x1-9,y1-9,x2+9,y2+9,p^);
  setfillstyle(1,5{darkgray});
  bar(x1-1,y1-1,x2+1,y2+1);

  setcolor(lightgray);
  rectangle(x1-2,y1-2,x2+2,y2+2);
  rectangle(x1-3,y1-3,x2+3,y2+3);

  setcolor(white);
  rectangle(x1-4,y1-4,x2+4,y2+4);
  rectangle(x1-5,y1-5,x2+5,y2+5);

  setcolor(lightgray);
  rectangle(x1-6,y1-6,x2+6,y2+6);
  rectangle(x1-7,y1-7,x2+7,y2+7);

  setcolor(darkgray);
  rectangle(x1-8,y1-8,x2+8,y2+8);
  rectangle(x1-9,y1-9,x2+9,y2+9);


  setfillstyle(1,lightblue);
  bar(x1+1,y1+1,x2-1,y1+27);

  settextstyle(1,0,1);
  setcolor(15);
  outtextxy(x1+(((x2-x1) div 2)-(textwidth(baslik) div 2 )),y1,baslik);
  setfillstyle(1,lightgray);
  setcolor(15);
  fare(goster);
```

```
end;

procedure pencerekapat(x1,y1,boy:integer);
begin
fare(gizle);
putimage(x1-9,y1-9,p^,normalput);
freemem(p,boy);
fare(goster);
end;
end.
```

ÖZGEÇMİŞ

1960 yılında Erzurumda doğdu. İlk, orta ve lise öğrenimini Erzurumda tamamladı. Kazım Karabekir Eğitim Enistitüsü Matematik bölümünden 1980 yılında mezun oldu, aynı yıl İstanbul Üniversitesi İktisat Fakültesini kazandı. Bu fakülteden 1984 yılında mezun oldu. Askerlik görevinin ardından, 1985 yılında Maliye Bakanlığı Gelirler Genel Müdürlüğüne denetim elemanı olarak girdi. Buradaki görevini 1988 yılına kadar sürdürdü ve bu yıl içerisinde, Atatürk Üniversitesi Bilgisayar Bilimleri Uygulama ve Araştırma Merkezinde Uzman olarak göreve başladı. Buradaki çalışmaları 1993 yılına kadar devam etti. 1993 yılında Atatürk Üniversitesi Erzurum Meslek Yüksekokulu Bilgisayar Programına Öğretim görevlisi olarak girdi. Halen bu göreve devam etmektedir.

1994 yılında "Atatürk Üniversitesi Bilgisayar Destekli Öğrenci İşleri Uygulaması Kayıt ve Not Takibi Alt Sistemi" adlı yüksek lisans tezi ile, Uzman oldu.