T.C.
MALTEPE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

SAYDAM KUTU TEST YÖNTEMİNİN NESNE TABANLI BİR SİSTEME UYGULANIŞI

Okan Vardarlı

Yüksek Lisans Tezi

Tez Danışmanı
Prof. Dr. Kemal KÖYMEN

İSTANBUL – 2008

T.C.
MALTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

SAYDAM KUTU TEST YÖNTEMİNİN NESNE TABANLI BİR SİSTEME UYGULANIŞI

YÜKSEK LİSANS TEZİ

Okan VARDARLI

Tez Danışmanı
Prof. Dr. Kemal KÖYMEN

İSTANBUL – 2008

Bu tez çalışması, Maltepe Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 14/3/2008 tarih ve 2008/10 sayılı kararıyla oluşturulan jüri tarafından ***Bilgisayar Mühendisliği Yüksek Lisansı Tezi*** olarak kabul edilmiştir.

JÜRİ

Prof. Dr. A. Kemal KÖYMEN
Danışman

Prof. Dr. Şaban EREN
Üye

Prof. Dr. İlhami YAVUZ
Üye

ÖZET

Yüksek Lisans Tezi, Saydam Kutu Test Yönteminin Nesne Tabanlı bir Sisteme Uygulanışı, T.C. Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı.

Genel bir tanımla, yazılımın test edilmesi, yazılım sistemlerinde hataların açığa çıkarılması sürecidir. Test etmek, hataları izleme ve düzeltme çabalarını içermez.

Yazılımın başarısız olmasına yol açabilecek pek çok değişik neden olabilir. Yazılımın tüm farklı başarısız olma biçimlerini saptamak genellikle olanak dışıdır. Çünkü test etmek, zaman ve paraya mal olur. Diğer taraftan test etmenin kapsamının (programın test edilen kısmının) maksimum olması önemlidir. Bu durum, özellikle hataların affedilemeyeceği kritik sistemler için doğrudur.

Test etmek için iki anahtar türde test yöntemi vardır: *Fonksiyonel Test* ve *Yapısal Test*. Bunlar sıklıkla *Kara Kutu Testi* ve *Saydam Kutu Testi* olarak anılırlar. Bu tezin amacı, saydam kutu test sürecini nesneye yönelik bir sistem üzerinde uygulamanın aşamalarını göstermektir. Bu çalışmada öncelikle test etme türleri ve teknikleri açıklanmış, program kodunu kullanan saydam kutu tekniği ayrıntılı olarak ele alınmıştır. Daha sonra nesneye yönelik test yöntemleri incelenmiş ve test programları yazılarak uygulama çalışması gerçekleştirilmiştir.

Bu tez 2008 yılında tamamlanmıştır ve 138 sayfadan oluşmaktadır.

Anahtar Kelimeler: Yazılım testi, yapısal test, saydam kutu testi, beyaz kutu testi, nesneye dayalı test.

ABSTRACT

Master Thesis, Applying Glass-Box Testing Strategy on an Object Oriented System, T.C. Maltepe University, Graduate School of Natural and Applied Sciences, Department of Computer Engineering.

In general, software testing is the process of uncovering evidence of defects in software systems. Testing does not include the debugging or repair of bugs.

Software can fail in many different ways. Detecting all of the different failure modes for software is generally infeasible. Because, testing costs time and money. On the other hand, it is important to maximize test coverage (the proportion of the program we have tested). This is especially true for critical systems where failures cannot be tolerated.

There are two key types of testing: Functional Testing and Structural Testing. These are often referred to as black-box and glass-box testing. The aim of this thesis is demonstrating the stages of glass box testing on an object oriented system. In this study, initially testing types and techniques are explained, and then glass box testing that uses program code is researched in detail. After this, object oriented testing approach is investigated and test cases are written to apply this knowledge.

This thesis has been completed in 2008 and it has 138 pages.

Keywords: Software testing, structural testing, glass box testing, white box testing, object oriented testing.

TEŞEKKÜR

Öncelikle uzun bir aradan sonra tekrar öğrenci olmaya beni yüreklendiren Maltepe Üniversitesi Fen Bilimleri Enstitüsü Müdürü Sayın Prof. Dr. İlhami YAVUZ'A, bu tez konumu seçerken beni yönlendiren, teşvik eden ve çalışmalarımda bilgi ve deneyimiyle bana ışık tutan danışman hocam Maltepe Üniversitesi Mühendislik Fakültesi Dekanı Sayın Prof. Dr. Kemal KÖYMEN'E, teknik konularda bilgi ve birikimlerinden faydalandığım, arkadaşlarım Sayın Emir AKAYDIN'A ve Sayın Fatih SÖKMEN'E ve kaynak sağlamadaki özverisinden ötürü Sayın Ali KAVURGACI'YA, dostluklarıyla bana güç veren ve çalışmalarımda bilgi ve birikimlerinden faydalandığım çalışma arkadaşlarım Maltepe Üniversitesi Endüstri Mühendisliği bölümünden Sayın Yrd. Doç. Dr. Çiğdem ALABAŞ USLU'YA ve Öğretim Görevlisi Sayın Dr. Nevin KARAARSLAN CENGİZ'E, evdeki çalışmalarımda emek ve manevi desteklerini esirgemeyen canım annem Sayın Leyla VARDARLI ve canım halam Sayın Emel VARDARLI'YA can-ı gönülden teşekkür ederim.

Annem ve Halama…

İÇİNDEKİLER

Sayfa

ŞEKİLLER

Sayfa

ÇİZELGELER

Sayfa

1. GİRİŞ

Yazılımın test edilmesi, yazılımda var olan ile istenen koşullar arasındaki farklılıkları (ki bu hatadır) bulmak üzere ve bu yazılım parçasının özelliklerini değerlendirmek için incelenmesi sürecidir [1]. Bir diğer tanımla yazılımın test edilmesi, bir program ya da sistemin hata bulmak amacıyla çalıştırılması sürecidir [2]. Ya da test, bir program ya da sistemin özelliklerinin veya yeteneklerinin istenen sonuçlara uygunluğunu belirlemek amacıyla yapılan herhangi bir etkinliği içerir [3]. Genel bir tanımla, yazılımın test edilmesi, yazılım sistemlerinde hataların açığa çıkarılması sürecidir [4].

Yazılım, girdilerin alındığı ve çıktıların üretildiği diğer fiziksel süreçlerden farklı değildir. Yazılımın farkı başarısız olma biçimindedir. Fiziksel sistemlerin çoğu sabit bir dizi biçimde (ve oldukça az sayıda) başarısızlığa uğrar. Bunun aksine, yazılım pek çok garip şekillerde başarısız olabilir. Yazılımın bütün farklı başarısız olma biçimlerini saptamak genellikle mümkün değildir [5].

Bir hata, geliştirme veya bakım sürecinin herhangi bir evresinde ortaya çıkabilir ve bir veya birçok hatadan, yanlış anlaşılmadan, ihmalden ya da programcıların yanlış değerlendirmesinden de kaynaklanmış olabilir. Test etmek yalnızca hata bulma çabalarını içerir; hataları izleme ve düzeltme çabalarını içermez. Bir diğer deyişle test etmek, hatayı irdeleme ve tamir etmeyi içermez.

Test etmek zaman ve paraya mal olur, ancak yazılımın test edilen kısmının olabildiğince geniş tutulması önemlidir. Şöyle ki kısıtlar olmasaydı kodun tamamının test edilmesi en idealidir. Bu özellikle hataların hoş görülemeyeceği kritik sistemler için doğrudur. Test kapsamını maksimum yapabilmek için bir dizi uygun test programları tasarlanmalıdır. Bir test programı öyle tasarlanmalıdır ki hata yakalama şansı yüksek olsun ve test ettiği sistemin hangi parçasını test ettiğinin amacı olsun. Bu programın kullanacağı girdiler ve programdan beklenen çıktılar belirtmelidir [6].

Test programının beklenen çıktılarını programca üretilen gerçek değerlerle kıyaslayacak bir beklenen değer kullanılabilir.

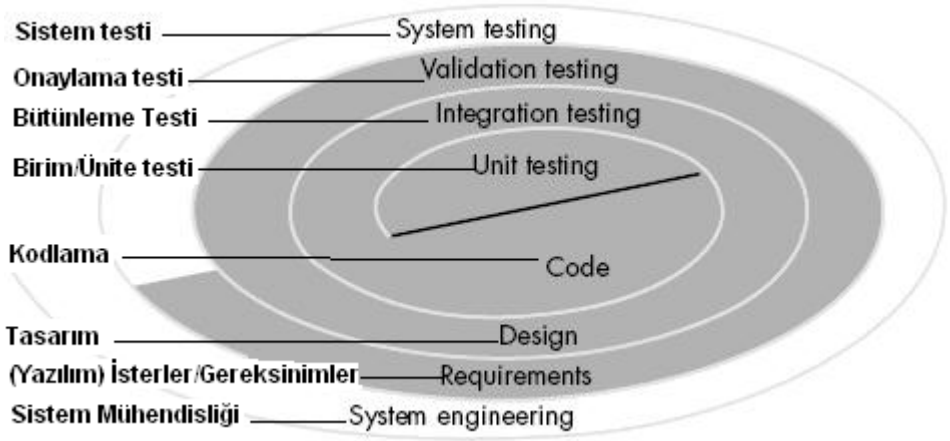
İki anahtar türde test yöntemi vardır: *Fonksiyonel Test* ve *Yapısal Test*. Bunlar sıklıkla *Kara Kutu Testi* ve *Saydam Kutu Testi* olarak anılırlar.

Fonksiyonel test, herhangi bir programın kendi girdi kümesindeki değerleri kendi çıktı aralığındaki değerlere dönüştüren bir fonksiyon olduğu bakış açısına dayanır [7]. Fonksiyonel testte programın bir kara kutu olduğu düşünülür. Test programları programın şartnamesine (specification) göre ve hiçbir kod yazılmadan tasarlanabilir. Fonksiyonel test ile programın şartnamede belirtilen tüm işlevlere sahip olup olmadığı kontrol edilebilir ancak programın istenmeyen bazı davranışları içerip içermediği kontrol edilemez.

Yapısal test, sisteme bir saydam kutu olarak bakar (kodu görebiliriz) ve test kodun çalıştırılmasına dayanır. Bu testin temel amacı, yazılımın cümlelerinin, kodun izleyebileceği bütün yolların, koşul cümlelerinin ve veri akışının doğruluğunun onaylanmasıdır [8]. Yapısal testte yerine getirilmeyen işlevler belirlenemez. İdeal olarak, bir yazılım test edilirken, belirtilen bütün özelliklerin yerine getirildiğinden ve bu özellikler için yazılan kodun doğru ve hatasız oluğundan emin olmak için hem fonksiyonel hem de yapısal test yöntemleri kullanılmalıdır.

Bir program üzerinde uygulanan testlerin birden çok seviyesi vardır ve bu seviyeler hem yordamsal hem de nesneye yönelik programlara uygulanabilir. Bu seviyeler aşağıdaki gibidir:

- Birim/Sınıf (unit) Testi
- Bütünleme (integration) Testi
- Onaylama (validation) Testi
- Sistem Testi



Şekil 1.1 - Sınama Stratejisi [9]

Birim testi, bir programın bireysel bileşenlerinin izole olarak test edildiği ilk uygulanan testtir. IEEE tanımı ise bireysel donanım ya da yazılım birimlerinin ya da birbiriyle ilişkili birim gruplarının test edilmesidir [10]. Nesneye yönelik sistemlerde, birim testi *Sınıf Testi* olarak adlandırılır ve bu tür test bu tezin temel odak noktasıdır.

Birim testinden yola çıkılarak testin bir sonraki evresi bütünleme testidir. Bütünleme testi, yazılım bileşenleri, donanım bileşenleri ya da her ikisinin birlikte test edilerek aradaki etkileşimi değerlendirmek üzere test edilmesidir. Birimler kendi başlarına doğru çalışıyorsa, birlikte de doğru çalışacaklar anlamına gelmez. Aşağıdan yukarı metoduyla en alt seviyedeki birimleri bütünleştirerek adım adım hiyerarşinin üstlerine doğru (sınıf diyagramı ya da modül ayrıştırma yöntemiyle) ilerlenir. Alt seviyedeki birimleri test etmek için bir test sürücüsü gereklidir [9].

Onaylama testi, isterlerle (requirements) uygunluğu doğrular. Bu test, "Doğru ürünü geliştirdik mi?" sorusunun yanıtını verir. Onaylama, isterlerle uygunluğu gösteren bir dizi kara kutu testlerinin yapılmasıyla başarılır.

Son olarak sistem testi, tamamlanmış, bütünleşmiş sistem üzerinde sistemin belirtilen isterlerle uygunluğunu test etmektir [10]. Sistem testi performans, güvenlik, denetlenebilirlik, düzenleniş (configuration) duyarlılığı, sistemi başlatma ve iyileştirilmesi için test etmeyi içerir [11]. Burada programcı veya test uygulayıcısı

tarafından yazılan test programlarının kullanılması yerine test verileri müşteri tarafından sağlanır, böylece programın normal işletme koşullarında çalıştığı gösterilir.

Bu çalışmada nesneye yönelik bir sistem üzerinde program kodları kullanılarak test süreci uygulamalı olarak en sade biçimiyle ele alınmıştır. Öncelikle test türleri ve teknikleri açıklanmış, bunlardan program kodunu kullanan saydam kutu tekniği daha ayrıntılı olarak anlatılmıştır. Daha sonra nesneye yönelik test yöntemleri incelenmiştir. *Test Uygulamaları* bölümünde testlere nasıl hazırlanıldığı örneklerle açıklanmıştır. Testlerin otomasyonunda *CPPUnit* [12] test aracından, dokümantasyonda da *Doxygen* [13] programından yararlanılmıştır. Bu yazılımlara ait dosyalar da çalışmanın sonunda listelenmiştir.

2. TEST TÜRLERİ VE TEKNİKLERİ

2.1. Saydam Kutu Testi

Saydam kutu testi [11], yapısal test, beyaz kutu testi, mantık güdümlü test [2] ya da tasarım tabanlı test [3] olarak da adlandırılan bu test yalnızca bir test yöntemi değil aynı zamanda *Test Tasarlama Yöntemi* gibi birçok test tekniğini açıklamada kullanılan bir isimdir. Bu tekniklerin en küçük ortak paydası bir yazılım mühendisinin belli bir parça yazılımı nasıl gördüğüdür.

Saydam kutu testinde, bir mühendis yazılımın içyapısına ilişkin bilgiyi kullanarak yazılımı gözden geçirir. Bu bilgi kullanılarak test verileri toplanır, test programları yazılır. Kısaca saydam kutu testi, modül düzeyinde uygulanmakta ve programın içyapısı denetlenmektedir.

Günümüzde saydam kutu testi pek çok alt kategorilere ayrılmaktadır. Hepsinin bir ortak noktası vardır, o da yazılım parçasını nasıl incelediklerdir. Daha yaygın olarak bilinen saydam kutu stratejilerinden bazıları şunlardır [8, 11]:

1. Cümle kapsama tekniği
2. Dal (basit koşul) kapsama tekniği
3. Birleşik koşul kapsama tekniği
4. Yol kapsama tekniği
5. Döngü kapsama tekniği

2.1.1. Cümle Kapsama Tekniği

Cümle kapsamanın hedefi yazılımdaki her bir cümleyi en az bir kere çalıştırmaktır. Bir test araç programı yardımıyla kodun ne kadarının kapsandığı ölçülebilir. Ancak bu teknik bir koşulun tüm olası sonuçlarının doğru çalıştığını garantilemez. Örneğin aşağıdaki kod parçasını ele alalım [14]:

```
if (s>1 && t==3)
   x=9;
```

Test programı: s=2, t=0.

Burada programcı bir hata yapmıştır, s>1 && t==3 yerine s>1 || t==3 yazılmak istenmiştir, ancak seçilen test verisi bu hatayı yakalayamaz.

2.1.2. Dal (basit koşul) Kapsama Tekniği

Dal kapsama, verilen bir parça kod içindeki her bir dalı test eden tekniktir. Örneğin, bir karar noktasındaki her bir mümkün dal, switch/case cümlesi en az bir kere çalıştırılmakta, böylece bütün ulaşılabilir kodun en az bir kez çalışacağı garanti edilmektedir. Burada her bir kararın doğru ve yanlış sonucu için en az birer test programı yazılmalıdır.

2.1.3. Birleşik koşul kapsama tekniği

Birleşik koşul kapsama tekniği, dal kapsama tekniğinin birden fazla koşulun birleştirilmiş halde genişletilmesidir. Burada koşullar birbirleriyle VE, VEYA ve DEĞİL mantıksal operatörleriyle birleştirilmiş hatta iç içe koşul cümleleri kullanılmıştır. Doğruluk tabloları kullanılarak koşulların her bir parçası için olası değerlerin permütasyonu oluşturulur. Koşul cümleleri karmaşıklaştıkça doğruluk tabloları da genişler.

İki koşul cümlesine sahip bir cümleyi ele alalım:
(Yaş > 18 VE Cinsiyet = K)
Bunun için doğruluk tablosu aşağıdaki şekilde olacaktır:

YAŞ	CİNSİYET	VE koşulu
= 18 Yanlış	= E Yanlış	= Yanlış
= 18 Yanlış	= K Doğru	= Yanlış
= 20 Doğru	= E Yanlış	= Yanlış
= 20 Doğru	= K Doğru	= Doğru

Çizelge 2.1

Böylesi bir koşul cümlesinin 100% kapsanması için dört farklı test yapmak gereklidir. Oysa basit koşul kapsama tekniğinde bir doğru bir de yanlış koşulun test edilmesi yeterli idi, o takdirde tablodaki diğer iki koşul test edilmemiş olurdu ve buradaki olası mantık hataları da gözden kaçabilirdi.

2.1.4. Yol Kapsama Tekniği

Bu teknik yapısal test tekniklerinin en güçlüsüdür ve verilen bir parça kod içindeki bütün yollarla ilgilenir. Bir diğer deyişle test süresince kodun her bir satırı en az bir kere ziyaret edilmelidir. Ancak ne yazık ki, pek çok satıra sahip bir yazılımda bunu başarmak çok zordur. Bu nedenle mühendisler bu tekniği programın küçük parçalarında ve yazılımın doğru çalışması için kritik olan kısımlarında kullanırlar [15]. Bu teknik ilk olarak McCabe tarafından önerilmiştir [9].

Bütün yolların kapsanması imkansız olduğuna göre, test programı sayısında nasıl bir kısıtlamaya gidilebilir? Bu konuda tanımlanmış bazı kıstaslar hangi yolların seçilmesi gerektiği konusunda bize yardımcı olurlar. Bu kıstaslar aşağıdaki gibi özetlenebilir [14]:

1. Lineer kod dizileri (linear code sequences)
2. Tüm tanım kullanım yollarının kapsanması (all-def-use-path coverage)
3. Döngüsel karmaşıklık ölçütü (cyclomatic complexity metric)

Yazılım mantığını ve kontrol yapılarını kullanarak test verileri üreten ve test bütünlüğünü irdeleyen metotlara alternatif metotlar da vardır ki bunlar kodun diğer özelliklerine odaklanırlar [5]. Bunlardan yaygın biçimde kullanılan bir yaklaşım koddaki değişkenlerin rollerine odaklanır. Diğeri ise hata tabanlıdır. Daha sonrakiler ise yazılımda değişiklikler yapmaya, değiştirilmiş sürümü test etmeye ve sonuçları karşılaştırmaya odaklanırlar.

2.1.4.1. Veri Akış Testi

Veri akış bilgisine göre test verisi üretmeyi tartışmak için, yazılımdaki değişkenlerin rolünü belirleyen bazı temel kavramlardan söz etmek gereklidir.

Bir değişkenin tanımlı olması bu değişkene bir değer atanması veya değişkenin değerinin değiştirilmesi ile gerçekleşir.
Örneğin izleyen cümlelerde

```
Y = 3 * X
Read (Y)
```

Y değişkenine yeni bir değer atandığında, Y değişkeni tanımlanmıştır. Bu durum veri akış çizelgesinde Y değişkeni için *def* şeklinde belirtilir.

Bir değişkenin kullanılmış olması onun bir cümle içinde yer alması ile gerçekleşir. Değişkenin değeri değişmemiştir. Rapps ve Weyuker tarafından daha ayrıntılı bir değişken kullanım tanımı verilmiştir [11]. Burada bir değişken için onun bir yüklem içindeki rolünü işaret eden yüklem kullanımı (*p-use*) tanımlanmıştır. Bir de hesabın bir parçası olarak değişkenin rolünü işaret eden hesaplama kullanımı (*c-use*) tanımlanmıştır. Her iki durumda da değişkenin değeri değişmemiştir.

Örneğin aşağıdaki cümlede

```
Y = 3 * X
```

X değişkeni kullanılmıştır. Burada c-use vardır. Diğer cümlede ise

```
if (X > 100)
   Y=max
```

X için *p-use* vardır.

Undefined gibi başka veri akış rolleri de vardır ancak bunlar izleyen bölümlerle ilişkili olmadığından kapsam dışı bırakılmıştır. Bazı belirli değişkenler için veri akış örneklerinin analizi hata yakalamada çoğu kez çok faydalıdır. Örneğin bir değişkenin tanımlanmadan kullanılması kodda bir hata göstergesidir. Değişkene henüz ilk değer ataması yapılmamıştır. Akıllı derleyiciler bu tür hataları yakalarlar. Programcılar ve

test uygulayıcıları değişken rol bilgisi belirleyebilen ve gösteren veri akış araçlarından yararlanabilirler. Bu araçlar deneticilerin işini kolaylaştırmak için kod izlemelerinden önce kullanılmalıdır.

Rapps ve Weyuker, veri akış açıklamalarını kullanarak ilgili kapsama amaçlarıyla eşleşen pek çok veri akış tabanlı test yeterlilik kıstasları belirlemişlerdir. Bunlar belli bir yol parçasını kullanan bir test kümesine aittirler, örneğin:
All def
All p-uses
All c-uses/some p-uses
All p-uses/some c-uses
All uses
All def-use paths
Bu kıstaslar içinde en güçlüsü *All def-use paths'*dir. Bu *all p-* ve *c-uses'ı* içerir.

Test uygulayıcısı, *all def-use* kıstasını sağlayabilmek için test altında olan yazılımdaki tüm var olan değişkenleri belirlemeli ve sınıflamalıdır. Çizelge şeklinde bir özet yararlıdır. Daha sonra her bir değişken için test verisi üretilir, böylece test süresince bütün değişkenlerin tanımı ve kullanımı deneyimlenmiş olur.

Saydam kutu testlerinin çoğunda olduğu gibi veri akış yaklaşımı da en çok birim testi seviyesinde verimlidir. Kod karmaşıklaştıkça ve düşünülmesi gereken daha çok değişken oldukça, veri akış rollerinin incelenmesi, yolların belirlenmesi ve testlerin tasarlanması test uygulayıcısının daha çok zamanını alır. Veri akış yönlü diğer problemler işaretçiler gibi dinamik olarak bağlı değişkenlerin kullanımı sırasında meydana gelir. Son olarak, veri akış testleri için, kontrol akış yönlü testlerde olduğu gibi güçlü bir destek sağlayan ticari programlar yoktur. Kodun hangi kısımlarının kapsanmadığını ve kapsama derecesini belirleyen araçlar da önemlidir. Veri akış yöntemleri için henüz bu programlar mevcut değildir.

2.1.5. Döngü Kapsama Tekniği

Döngüler en çok kullanılan kontrol yapıları arasındadır. Deneyimli yazılım mühendisleri pek çok hatanın döngü yapısıyla bağlantılı olduğunun farkına varırlar. Bunlar genellikle kötü programlama alışkanlıkları ve denetimlerin olmaması yüzündendir. Bu nedenle, test sırasında döngülere özellikle dikkat edilmelidir.

Beizer döngüleri dört farklı kategoriye ayırır:
Basit, iç-içe, birbirine bağlı ve yapısal olmayan döngüler [11].

Beizer, eğer kullanılan programda yapısal olmayan döngüler varsa bunların yapısal programlama teknikleriyle düzeltilmesini önerir. Böylece test uygulayıcıları diğer döngü çeşitleriyle ilgilenebilirler.

2.2. Kara Kutu Testi

Kara kutu testi yazılımın işlevsel gereksinimlerini test eder. Bu yöntemde daha önce de söz edildiği gibi programın bir kara kutu olduğu düşünülür. Test programları programın isterlerine göre ve hiçbir kod yazılmadan tasarlanabilir.

2.3. Gri Kutu Testi

Bu yöntem saydam kutu testi ile kara kutu testinin birleşmesiyle ortaya çıkmıştır. Kara kutu ve gri kutu testleri bu tezin konusu olmadıklarından daha fazla ayrıntıya yer verilmemiştir.

3. NESNEYE YÖNELİK TEST

Bu tezde, nesneye yönelik yaklaşım, yaygın kullanımı ve avantajları nedeniyle seçilmiştir. Bu avantajlar şu şekilde özetlenebilir:

- Nesneye yönelik model, nesneler ve bunların birbiriyle etkileşimleri çerçevesinde modellenmiştir. Bu yöntemde ilgi alanları birbirinden ayrılır, bu da problem tanım kümesinin daha iyi anlaşılmasına yol açar.
- Hemen hemen bütün nesneye yönelik yaşam döngüsü modelleri yaşam döngüsü boyunca aynı biçimde bir yapı sunar. Bu yapı, isterler analizi safhasında soyut iken tasarım ve uygulama safhasında daha somut ve ayrıntılı maddelere dönüşür.
- Bakım safhasında nesnelerin izlenmesi ve işlemlerin yerel etkisi daha kolaylaşmıştır. Buna ek olarak, kalıtım (inheritance) ve çok biçimlilik (polymorphism) nedeniyle sisteme eklemeler düzenli olarak daha az bir çabayla gerçekleşir.
- Hemen hemen bütün nesneye yönelik modeller var olan kodun tekrar kullanımını desteklerler.

Nesneye yönelik test stratejisi, geleneksel yöntemlere benzer ancak bazı taktik farklılıkları vardır [9]. Nesneye yönelik analiz ve tasarım modelleri yapısal ve içerik olarak benzerdirler, bu nedenle test süreci bu modellerin incelenmesi ile başlar. Kod yazıldığı zaman, nesneye yönelik test küçük çapta sınıf testleriyle başlar. Sınıf operasyonlarını (metot veya servisler) çalıştırmak ve bir sınıfın diğerleriyle etkileşiminde hatalar olup olmadığını incelemek üzere bir dizi test tasarlanır. Sınıflar alt sistemler oluşturmak üzere bütünleştirildikçe, diğer sınıflarla etkileşimlerini tam olarak ortaya koyan iş parçacığı tabanlı (thread-based), kullanış tabanlı (use-based), ve grup tabanlı (cluster based) ve hata tabanlı (fault based) yaklaşımlar uygulanır. Son olarak, nesneye yönelik analiz modelinin bir parçası olarak geliştirilen kullanım senaryoları (use cases), yazılım onaylama seviyesinde hataları otaya çıkarmada kullanılır.

Yazılım geliştirme ve testteki eski ve yeni yöntemler arasındaki fark, girdileri çıktılara dönüştüren fonksiyonlara odaklanmak yerine nesnelere odaklanmaktan daha

derindir [4]. En göze çarpan fark, nesneye yönelik yazılımın, bir problemi bir dizi nesneler halinde modellemesi ve daha sonra nesnelerin birlikte çalışarak bir çözüm üretmek üzere tasarlanmasıdır. Bu yaklaşımın dayandığı kavram, bir probleme bulunan çözümün zaman içinde değişebildiği gibi, problemin yapısı ve bileşenlerinin bu sıklıkta ve oranda değişmediğidir. Sonuç olarak, bir probleme göre yapısı tasarlanan bir program, daha sonraki değişikliklere daha kolay adapte olacaktır. Problem ve bileşenlerini bilen bir programcı, bunları yazılım içinde tanıyabilir ve böylece programın bakımı daha da kolaylaşır. Bundan başka, bileşenlerin oluşumunda problemden yola çıkıldığından, benzer ya da ilgili bir başka problemin çözümünde geliştirilecek programlarda bu bileşenler kullanılabilir, böylece yazılım parçalarının tekrar kullanımı sağlanmış olur.

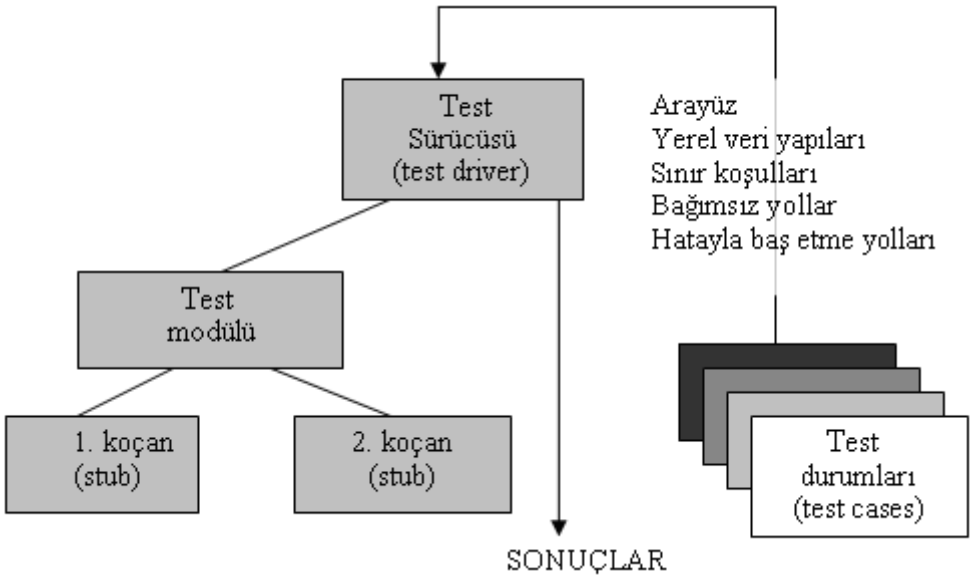
Geleneksel yöntemlerle geliştirilmiş projeler ile nesneye yönelik teknolojilerle geliştirilmiş projeler arasındaki bir diğer fark, yazılımın hedefleridir. Örneğin pek çok firma için önemli bir yeni amaç, genişleyebilen bir tasarıma sahip, tekrar kullanılırlığı olan hatta tekrar kullanılırlığa sahip bir tasarımla nesne-tabanlı çerçevede yazılım üretmektir. Test, bu amaçları yerine getirirken hataları ortaya çıkarmalıdır. Geleneksel test yaklaşımları ve teknikleri bu amaçları hedeflemezler.

3.1. Birim Testi

Nesneye yönelik yazılım düşünüldüğünde birim kavramı değişir [9]. Sarmalamak (encapsulation) sınıf ve nesne tanımlarını belirler. Bunun anlamı, her bir sınıf ve her bir sınıfın bir örneği olan nesnenin, veri üzerinde uygulanan nesne niteliklerini ve operasyonları içerdiğidir. Tek bir modülün test edilmesindense, test edilebilir en küçük birim sarmalanmış (kapsülleştirilmiş) sınıf ya da nesnedir. Bir sınıfın birden çok operasyonları olabilir ve bir operasyon da birden fazla sınıf tarafından kullanılabilir, bu birim testinin anlamını dramatik olarak değiştirir.

Birim testlerinin genel çevresi Şekil 3.1'de gösterilmiştir. Test sürücüsü (test driver), program verisini alıp test edilecek modüle ileten ve test sonucunu yazan bir ara

programdır. Koçan (stub) ise bir kukla (dummy) alt program ve test edilen modülün özet bir versiyonu olup, bu modülün ara birimi olarak kullanılmaktadır.



Şekil 3.1 - Birim Test Çevresi [9]

Geleneksel birim testlerinde olduğu gibi nesneye yönelik testlerde de bir operasyon izole biçimde test edilemez, ancak bir sınıfın parçası olarak test edilebilir. Bunu canlandırmak için şöyle bir örnek verebiliriz: X operasyonunun (C++'da bir metot) süper sınıf için tanımlandığını ve birtakım alt sınıflar tarafından kalıtımla alındığını düşünelim. Her bir alt sınıf X operasyonunu kullanır, ancak bu kullanım alt sınıf için tanımlanmış operasyonlar ve gizli (nesneye özel) nitelikler çerçevesinde uygulanır. X operasyonunun kullanımı farklılaşabildiğinden, bu operasyonun her bir alt sınıf için oradaki tanım çerçevesinde test edilmesi gereklidir. Bunun anlamı, geleneksel birim test yönteminde olduğu gibi bir operasyonun bağımsız testi nesneye yönelik kapsamda etkili değildir.

Nesneye yönelik yazılımlar için sınıf testi, geleneksel yöntemde birim testlerine karşılık gelir. Geleneksel yazılımların bir modülün algoritmik ayrıntılarına ve modül ara yüzündeki veri akışına odaklanan birim testlerinin aksine, nesneye yönelik sistemlerde sınıf testleri, sınıf tarafından sarmalanmış operasyonlar ve sınıfın durumsal davranışlarına göre belirlenir. Sınıf testleri ilerleyen bölümlerde daha fazla açıklanacaktır.

3.2. Bütünleme Testi

Nesneye yönelik programlamada hiyerarşik bir kontrol yapısı olmadığından, geleneksel yukarıdan-aşağı ve aşağıdan-yukarı bütünleme stratejilerinin pek anlamı yoktur. Bunun yanında, operasyonları teker teker sınıfa bütünleştirmek de bileşenlerin doğrudan ya da dolaylı olarak etkileşiminin bir sınıfı oluşturması nedeniyle çoğu kez olanaksızdır [9].

Nesneye yönelik sistemlerin bütünleştirilmesinde iki farklı strateji vardır [16]. İlki iş parçacığı tabanlı test, sistemdeki bir girdiye ya da olaya cevap vermesi gereken bir dizi sınıfı bütünleştirir. Her bir iş parçacığı tek tek bütünleştirilir ve test edilir. Yan etkilerin olmaması için bağlanım (regression) testi uygulanır. İkinci bütünleme yaklaşımı, kullanış tabanlı test, sistemi kurmaya servis sınıflarını kullanmayan veya çok az kullanan *bağımsız sınıflar* olarak adlandırılan sınıfların test edilmesi ile başlar. Bağımsız sınıflar test edildikten sonra bir sonraki aşama, bağımsız sınıfları kullanan *bağımlı sınıfların* test edilmesidir. Bu bir dizi bağımlı sınıf katmanlarının test edilmesi, bütün sistem kurulana kadar devam eder.

Nesneye yönelik yazılımın bütünleme testlerinde uygulanan bir diğer strateji ise tek adımlık grup (cluster) testidir [17]. Burada çevrimsel artıklık kodlaması (cyclic redundancy check) ve nesne-ilişki (entity relationship) modeli kullanılarak belirlenen, ortaklık yapan bir grup sınıf, bu ortaklıkta oluşabilecek hataları ortaya çıkarmak üzere tasarlanan test programları kullanılarak çalıştırılır.

3.3. Onaylama Testi

Onaylama ya da sistem düzeyinde bir sınıfın bağlantı ayrıntıları görülemez. Geleneksel onaylamada olduğu gibi, nesneye yönelik sistemlerin onaylanmasında da odak noktası kullanıcının görebildiği faaliyetler ve kullanıcının tanıyabildiği sistem çıktılarıdır. Onaylama test programlarının hazırlanmasına yardımcı olmak için, test uygulayıcısı analiz modelinin bir parçası olan kullanım senaryolarından

yararlanmalıdır. Kullanım senaryosunun kullanıcı etkileşim isterlerindeki bir hatayı ortaya çıkarması oldukça büyük bir olasılıktır.

Onaylama testlerinde geleneksel kara kutu test yöntemi de kullanılabilir. Buna ek olarak test programlarının hazırlanmasında, nesneye yönelik analizin bir parçası olarak yaratılan olay akış şeması ve nesne davranış modelinden yararlanılabilir.

3.4. Sınıf Testi

Sınıf testinin nesneye yönelik programlar için birim testi olduğunu düşünürsek, birim kavramını yeniden tanımlamamız gerekir. Yapısal dillerde birim, bir alt yordam ya da fonksiyon gibi ele alınmıştı. Nesneye yönelik dillerde metot alt yordama benzetilebilir ve bu benzetme birimin orijinal tanımına da uyar: bir metot tek bir görev yerine getirir ve tek bir programcı tarafından yazılmalıdır. Metotlar kendi görevlerini yerine getirebilmek için çoğu kez diğer metotlara dayanırlar ve sıklıkla bir sonrakini testten önce bir dizi metodun çalıştırılması gerekebilir. Kısaca bir metot bir birim olarak düşünülebilir [7].

Nesneye yönelik programlama, sınıf kavramını da beraberinde getirir. Sınıf bir nesnenin özelliklerini tanımlar. Nesnenin nitelikleri (değişkenler) ve davranışları (metotlar ya da operasyonlar) vardır. Sınıfın her bir örneğine nesne adı verilir. Sınıf, aynı zamanda test edilebilir bir birim olarak da görülebilir. Ancak bu kalıtımın çok az kullanıldığı ve alt sınıfların (diğer sınıflardan türetilen sınıflar) izole bir biçimde test edildiği yerlerde uygulanabilir [7].

Nesneye yönelik sistemlerin testinde metodu ya da sınıfı bir birim olarak seçmek gerekli değildir. Bazı durumlarda birini tercih etmek daha uygun olabilir [7], başka durumlarda ise bunların birleşimini kullanmak gerekebilir. Sınıfı bir birim gibi düşünsek de gene de onun içindeki metotları teker teker test etmemiz gereklidir. Bir diğer deyişle bir metodu tek başına test etsek de metotlar sınıflardan ayrı var olamazlar. Sınıf testi, içerdiği tüm metotların iş birliği içinde çalıştıklarını ortaya koymalıdır [18].

Bir sınıf yüksek düzeyde sarmalandığından test ederken de kara kutu testinin uygulanması beklenebilir. Ancak sınıf testinin bu şekilde yapılmasının doğru olmadığı bazı kaynaklarda belirtilmiştir [19]. Bu nedenle saydam kutu testi ve kara kutu testinin beraber kullanılması en uygunudur [20].

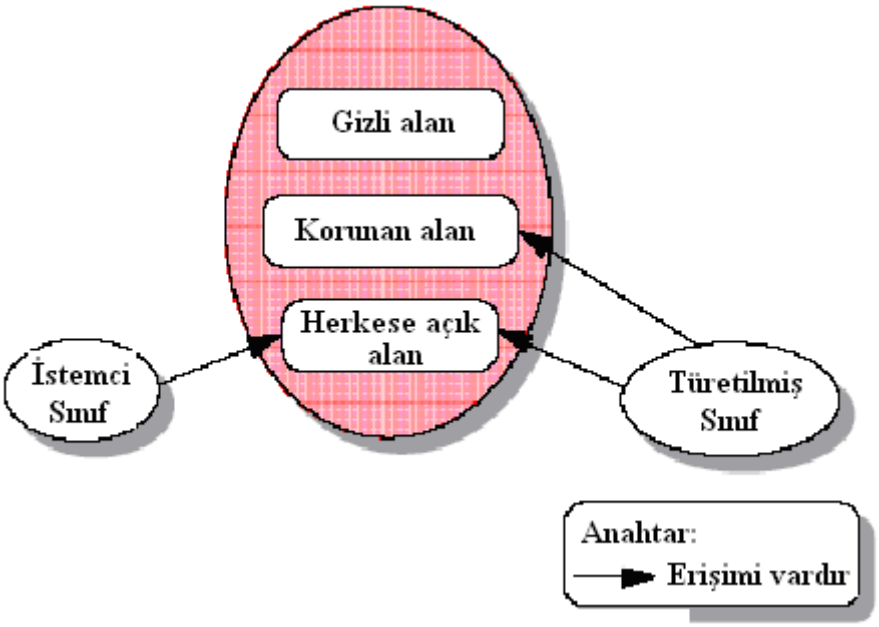
Geleneksel saydam kutu yaklaşımı kontrol ya da verinin akışı analizine dayanır [16]. Genel kanı bu yöntemlerin metot testi için yeterli olduğu ancak sınıf testi için yetersiz olduklarıdır. Geleneksel akış çizgesi (flow-graph) yaklaşımlarının bu alanda uygulanabilirliği çeşitli kaynaklarca sorgulanmıştır. Akış çizgesi yaklaşımının nesneye yönelik anlayışla tutarsız olduğu ya da metot düzeyinde kontrol hatalarına pek fazla rastlanmayacağı tartışılmıştır. Sınıf testinin daha kapsamlı ele alınması için kalıtımın var olduğu durumlar göz önüne alınmalıdır.

3.4.1. Sınıf Tanımları

Sınıf tanımı farklı programlama dillerinde temel bir şablon izler: özniteliklerin (features) tanımı ve sınıfın veri temsili. Fonksiyonlarına göre sınıf öznitelikleri temel olarak altı grupta toplanır [22] :

1. Create (Constructors)
2. Destroy (Destructors)
3. Modifiers
4. Predicates
5. Selectors
6. Iterators

Sınıfın farklı kısımlarının tanımı, programcının dikkatli bir şekilde nelerin dış dünyanın erişimine açık nelerin de gizlenmesine karar vermesiyle farklı bilgi saklama düzeyleri ortaya koyar. Programlama dilleri çeşitliliğini düşünerek (C++, Eiffel, Smalltalk, Self, Simula gibi), üç alan açık olarak tanımlanabilir. Her biri de artan düzeyde bilgi saklar [Şekil 3.2].



Şekil 3.2 – Bir Sınıfın Üç Alanının Özeti [22]

Gizli Alan:

Bu alanda maksimum bilgi saklama sağlanır. Buraya yalnızca sınıfın kendi üyelerinin öznitelikleri erişim hakkına sahiptir; türetilmiş (derived) sınıfların özniteliklerinin dahi erişim hakkı yoktur. Bu düzeyde bilgi saklama, istenmeyen ulaşımı engellemiş olur.

Korunan Alan:

Bir sınıfın korunan alanının iki görüntüsü vardır: istemci (client) sınıfından görüntüsü ve türetilmiş sınıftan görüntüsü. İstemci sınıfından görüntülendiğinde gizli alanla aynı erişim kısıtlarına sahiptir. Ancak türetilmiş sınıftan görüntülendiğinde kısıtlar kalkar. Böylece türetilmiş sınıfa bu alanda tanımlı öznitelik ya da veri üyelerine tam erişim hakkı verir.

Herkese Açık Alan:

Adından da anlaşıldığı gibi tüm sınıflar bu alanda tanımlı öznitelik ya da veri üyelerine erişim hakkına sahiptir. Bilgi saklamanın yoksunluğu bilgiyi dışarıdan gelebilecek müdahalelere karşı korunmasız kılar. Bu nedenle bu alan, sadece sınıfın dış ara yüzlerinin tanımı için kullanılmalıdır.

Bu alanlarda, nesnenin durumunu değiştirmeyen özniteliklerin testinin öncelikle yapılarak güvenin artırılacağı önerilmektedir. Bu, nesnenin durumunu değiştirmeyen özniteliklerin kodunun da basit olacağı varsayımına dayanmaktadır. Daha doğru bir uygulama en basit (en az kod karmaşıklığına sahip) koda sahip özniteliğin en önce test edilmesi şeklindedir.

En basiti kodun satır sayısını ölçen tipte olanından McCabe'in döngüsel karmaşıklık ölçütüne kadar pek çok kod karmaşıklığını ölçen ölçütler vardır. Bu ölçütler, sınıfın belli bireysel özniteliklerinin karmaşıklığını ölçmede kullanılırlar ve test uygulayıcısına, kodun hangi özniteliklerinin önce test edilmesi gerekliliğini belirlemesinde doğru yolu gösterirler. Yukarıda açıklanan sınıf öznitelikleri gruplarının ve alan tanımlarına göre taban sınıflarının (base class) test edilme yöntemi Turner ve Robson [22] tarafından ayrıntılı olarak açıklanmıştır.

3.4.2. Kalıtım

Sınıflardan, kalıtım adlı ilişki kullanılarak yeni sınıflar ya da alt sınıflar tanımlanabilir. Kalıtım, sınıflar üstünde hiyerarşik bir organizasyondan yararlanır ve bir alt sınıfın üst sınıfın niteliklerini kalıtımla almasına, genişletmesine, kısıt getirmesine ya da yeniden tanımlamasına izin verir. Böylece alt sınıflar, üst sınıfın sahip olmadığı bazı niteliklere sahip olabilirler [21].

Kalıtım, sınıf testinde bazı güçlüklere sebep olur. İlk olarak daha önce bir taban sınıfın test edilmesine karar verilebilir. Daha sonra da koçanlar kullanarak türetilmiş sınıflar test edilebilir. Dolayısıyla, alt sınıfların, kalıtımla aldıkları nitelikler ve operasyonları üst sınıfta test edildikleri halde yeniden bütün olarak test edilmesidir. Kısaca daha önce çalıştırılmış test programlarının bir diğer sınıf için ortak kullanımı mümkün değildir [21].

Burada sorulması gereken, bir alt sınıf üzerinde ne kadar test uygulanması gerektiğidir. Bu, daha ayrıntılı iki soruya yol açar [18]:

- Kalıtımla alınan metotlar test edilmeli midir?
- Üst sınıf testleri kalıtımla alınan ve yeniden tanımlanan (overriding) metotlarda kullanılabilir mi?

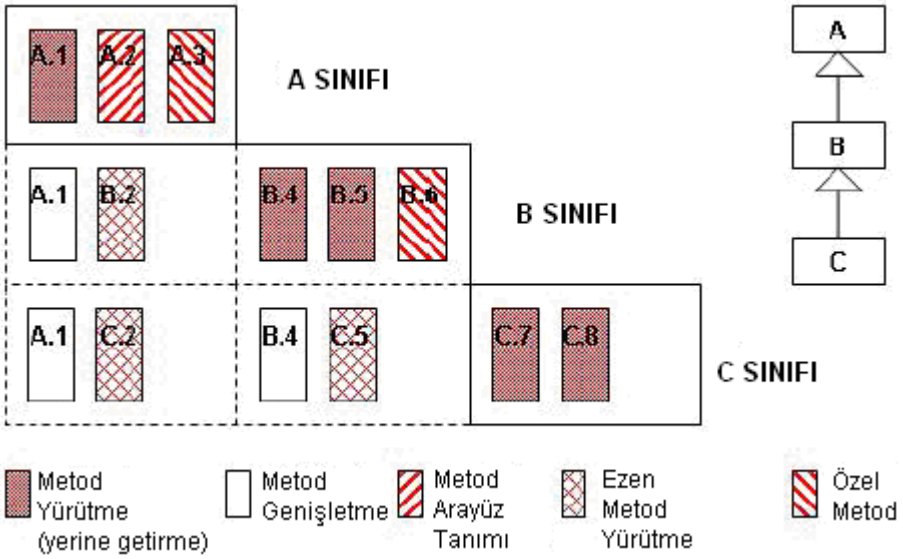
Kalıtım içeren sınıf testi programlarında bu soruları hedef alan iki ana teknik vardır:

- Düzleştirilmiş sınıflar ve
- Artımlı sınıf testleri

3.4.3. Sınıf Testi Teknikleri

3.4.3.1. Düzleştirilmiş Sınıf Testi

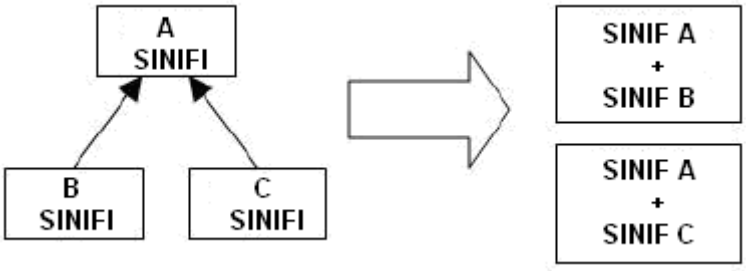
Düzleştirilmiş bir sınıf, test edilen alt sınıf ile üst sınıfın tek bir sınıf haline getirilmiş halidir. Bu kavram Şekil 3.3'de gösterilmektedir.



Şekil 3.3 - Düzleştirilmiş sınıflar [18]

A sınıfının üç tane metodu vardır, bunlardan biri özeldir, kalıtımla alınamaz. B sınıfı ise A sınıfının bir alt sınıfıdır ve A.1 metodunu kalıtımla almıştır. B sınıfı üç metoda daha sahiptir, bunlardan biri özel tiptedir. C sınıfı ise B sınıfının bir alt sınıfıdır ve A.1 ve B.4 metotlarını kalıtımla almıştır, B.2 ve B.5 metotları ise yeniden tanımlanmıştır. Bunlardan başka C'nin kendi metotları da vardır.

Kalıtım hiyerarşisi büyüdükçe bu düzleştirme tekniği daha da karmaşık bir hal alır. Aynı zamanda üst sınıfta uygulanan test alt sınıfta kullanılmaz ve böylece test iki kere yapılmış olur [21]. Bir sınıfın birden çok alt sınıfı olduğunda (Şekil 3.4'de görüldüğü gibi) sorun daha da büyür. Böylece daha çok sayıda düzleştirilmiş sınıfa sahip oluruz, dolayısı ile daha çok test tekrarı ortaya çıkar.



Şekil 3.4 - Düzleştirilmiş sınıflar

Düzleştirilmiş sınıfa ilişkin bir diğer sorun, bu sınıfın düzleştirilmiş olması zorunluluğu olmadığıdır. Sınıflar asıl kullanılacağı şekilde test edilmemektedir, uygulamanın son şeklinde sınıflar düzleştirilmeyecektir. Bunun anlamı testin bu aşamasında sınıf etkileşimi nedeniyle oluşabilecek hataların bulunmayabileceğidir.

3.4.3.2. Artımlı Sınıf Testi

Artımlı sınıf testi tekniği referanslarda ayrıntılı olarak ele alınmıştır [18, 23]. Öncelikle sınıf hiyerarşisinde en üstteki sınıfın (taban sınıf) bütün nitelik ve metotları ayrıntılı olarak test edilir. Test edilen her bir niteliğin bir test geçmişi (tarihçe) vardır. Taban sınıfın bir alt sınıfı yaratıldığında, bu sınıf kalıtımla nitelik ve metotları aldığı gibi test geçmişini de alır.

Kalıtımla alınan test geçmişi, üretilen sınıf için bir test seti hazırlanırken temel olarak kullanılır. Kalıtım içeren metotlarda, testlerde olduğu gibi test geçmişleri de süper sınıfta olmayan metotların testlerini içermek üzere güncellenmelidir.

Her ne kadar kalıtımla alınan metotların testi tekrar bir test gibi görünse de, gerçekte minimumdur, uygun olduğu yerlerde üst sınıf test setinden test programları alınıp alt sınıfın testlerinde kullanılabilir.

Artımlı sınıf testinin avantajı üst sınıf test setinden test programlarının alınıp alt sınıfın testleri için uygun bir test seti hazırlanmasında kazanılan zamandır. Yalnızca yeni ya da yeniden düzenlenen metotlar için yeni testler yazılır.

3.4.4. Test Düzeylerinin Özeti

Nesneye yönelik programlama genellikle aşağıdan yukarı doğru dört seviyede uygulanır [6,24]:

a) Metot seviyesinde test etmek bir sınıfın her bir metodunun teker teker test edilmesi demektir.
b) Metotlar ve nitelikler bir sınıfı oluşturur. Sınıf seviyesinde (ya da sınıf içinde) test, bir sınıfı oluşturan bileşenler arasındaki etkileşimin test edilmesi demektir.
c) Birbiriyle işbirliği yapan nesnelerin ait oldukları sınıflar bir demeti (grup, cluster) oluşturur. Demet-seviyesinde (ya da sınıflar arası) test, nesneler arasındaki etkileşimin test edilmesi demektir.
d) Demetler bir sistemi oluşturur. Sistem düzeyinde test, bir sistemin kullanıcıya görünen dış girdi ve çıktılarıyla ilgilidir.

Süper sınıfın test edilmesi:

- Her bir metot ve etkileşimin test edilmesi

Alt sınıfın test edilmesi:

- Kalıtımla gelen metotlar: Çok az miktarda tekrar test etmeye gerek vardır.
- Üst sınıfta test edilmiş, tanımı ve uygulamasında (implementation) değişiklik yapılmamış metotların tekrar test edilmesine gerek yoktur.
- Bütünleme testleri gerekli olabilir.
- Yeniden tanımlanan (override) metotlar ayrıntılı test edilmelidir.
- Tekrar kullanım değişikliğe bağlıdır.

Sınıflar arası test:

Kirani ve Tsai [25] bir seri çoklu sınıf rasgele test programları geliştirmek için aşağıdaki adımları önerirler:

- Her bir istemci sınıfı, bir seri rasgele test programları geliştirmek için sınıf metotlarının bir listesini kullanır. Metotlar diğer sunucu sınıflarına mesajlar gönderirler.
- Yaratılan her mesaj için katılımcı sınıf ve sunucu nesnedeki karşılık gelen metot belirlenir.
- Sunucu nesnedeki her bir metot için (istemci sınıfından gönderilen mesajlar tarafından çağrılan), bu metot tarafından gönderilen mesajlar belirlenir.
- Her bir mesaj için, bir sonraki düzeydeki metotlar belirlenir ve bunlar bir test sırasına sokulur.

3.4.5. Kalıtım Hataları

Özellikle nesneye yönelik sistemlerde test konusunda yapılan çalışmalar, sınıf testlerinde kalıtımla ilgili bir dizi hata aramayı önerir [18]. Bunlar aşağıda sunulduğu gibidir:

- Yanlış ilk değer atama: Süper sınıf ilk değer ataması yapılmamıştır ya da yanlıştır.
- Yerine getirilmeyen yeniden tanımlama: Bir süper sınıf metodunun bir alt sınıfa özel uygulaması atlanmıştır, yani, süper sınıf metodu, alt sınıf nesnesine metodun davranışı o nesneye uygun olmadığı halde yanlış bir şekilde bağlanmıştır.
- Şeffaf erişim: Bir alt sınıf içindeki bir süper sınıf değişkeni görülebilir (izlenebilir) ve alt sınıf metotları doğrudan bu değişkenleri güncelleyebilir. Bu evrensel değişkenlere sınırsız erişim hakkı tanınmasıyla aynı tipte bir sorundur. Değişkene alt sınıf tarafından yapılan değişiklikler, süper sınıfa istenmeyen bir darbe etkisi yapar.
- Spagetti kalıtım: Çoklu kalıtım (bir alt sınıfın iki veya daha çok üst sınıftan miras alması) ya da derin hiyerarşi kalıtımı hatalara sebep olacak şaşkınlığa yol açabilir.

- Garip kalıtım: Kodu bir diğer sınıfla paylaşmak için kalıtım kullanılır. Bu kolaylık kalıtımı diye adlandırılır ve yapılması pek hoş karşılanmaz.
- Şişman kalıtım: Bir alt sınıf süper sınıftan kullanımına uygun olmayan bir metodu kalıtımla almıştır.

3.5. Test Programı Tasarımı

Nesneye yönelik yazılımlar için test programı yöntemleri halen gelişmektedir. Nesneye yönelik test program tasarımına genel kapsamlı bir yaklaşım Berard tarafından tanımlanmıştır [9]:

1. Her bir test programı benzersiz olarak belirlenmeli ve test edilecek sınıfla açık bir biçimde ilişkilendirilmelidir.
2. Testin amacı belirtilmelidir.
3. Her bir test için test adımları belirlenmeli ve her test aşağıdakileri içermelidir:
   a. Test edilecek nesnenin belirlenen durum değerleri listesi
   b. Test sırasında uygulanacak mesajların ve metotların listesi
   c. Nesne test edilirken oluşabilecek kısıtlar
   d. Dış koşulların listesi (testi uygulayabilmek için yazılımın bulunduğu dış çevrede yapılan değişiklikler)
   e. Test etmeyi anlamak ve uygulamaya yardımcı olan ek bilgiler

Yazılımın girdi-süreç-çıktı görünümü ya da modüllerin algoritmik ayrıntıları tarafından belirlenen geleneksel test programı tasarımından farklı olarak nesneye yönelik test, bir sınıfın durumlarını deneyen bir dizi metotların tasarımına odaklanır.

3.6. Test etmeyi ne zaman bitirmeliyiz?

Test süreci potansiyel olarak sonsuzdur. Bütün hatalar ortaya çıkarılana ve yok edilene kadar test edemeyiz – bu mümkün değildir.

Saydam kutu testinde amaç programın iç bileşenlerinin doğru çalıştığından emin olmaktır [5]. Genel odak noktası program cümleleri ve dalları gibi yapısal

elemanlardır. Test uygulayıcısı program yapısında hata olup olmadığını belirlemek için bu yapısal elemanları kullanan test programları geliştirir. Test uygulayıcısı seçilen bütün yapısal elemanların kullanılmasıyla hataların ortaya çıkarılmasını umar.

Test uygulayıcısı, testin odak noktasının hangi yapısal elemanlar olacağına, uygun test verisinin seçilmesine ve yazılımın yeterince doğru çalışması amacıyla yapılan testleri sonlandırmaya karar vermek için bir çerçeveye gereksinim duyar. Resmi olarak test veri yeterlilik kıstası durdurucu kuraldır [26]. Bu kurallar yeterli miktarda test yapılıp yapılmadığını belirlemekte kullanılabilir. Kıstaslar bir programın test edilmesinde minimum standartlar gibi görülebilir. Yeterlilik kıstaslarının uygulama kapsamı aşağıdakileri içerir:

i) Test boyunca odaklanılacak program özelliklerinin seçilmesinde test uygulayıcısına yardımcı olmak
ii) Seçilen özelliklere göre test veri seti seçmede test uygulayıcısına yardımcı olmak
iii) Test için nicel amaçlar geliştirmede test uygulayıcısını desteklemek
iv) Bir program için testin durdurulup durdurulmayacağı konusunda test uygulayıcısına ipucu vermek

Bir program, seçilen kıstaslara göre verilen bütün (hedeflenen) yapısal elemanlar test edildikten sonra yeterince test edilmiştir denebilir.

Gerçekçi olarak test, bütçe, zaman ve kalite arasında bir pazarlıktır. Kar modelleri tarafından belirlenir. En karamsar ve ne yazık ki en çok kullanılan yaklaşım, ayrılan kaynaklardan biri – zaman, bütçe ya da test programları – tükendiğinde testi durdurmaktır. İyimser durdurma kuralı ise ya güvenilirlik isterleri karşılanıyorsa ya da daha fazla test etmeye devam etmenin masrafı elde edilecek faydaya değmeyecek ise test durdurulmalıdır [27]. Bu genellikle, güvenilirlik modelleri kullanılarak test edilen yazılımın güvenilirliğinin öngörülmesini ya da değerlendirilmesini gerektirir. Her bir değerlendirme tekrar tekrar şu döngüyü gerektirir: hata verisi toplama, modelleme ve değerlendirme. Bu model çok güvenilir sistemler için uygun değildir çünkü gerçek uygulama alanındaki hata verisini toplamak çok uzun sürer.

4. SINIF TESTİ ÜZERİNE BİR UYGULAMA

Uygulama projesi olarak Stephen R. Schah'ın kitabı "Object-Oriented Classical Software Engineering" in 6. baskısı Appendix A'da yer alan "Ophelia' Oasis Otel Otomasyonu" ele alınmıştır.

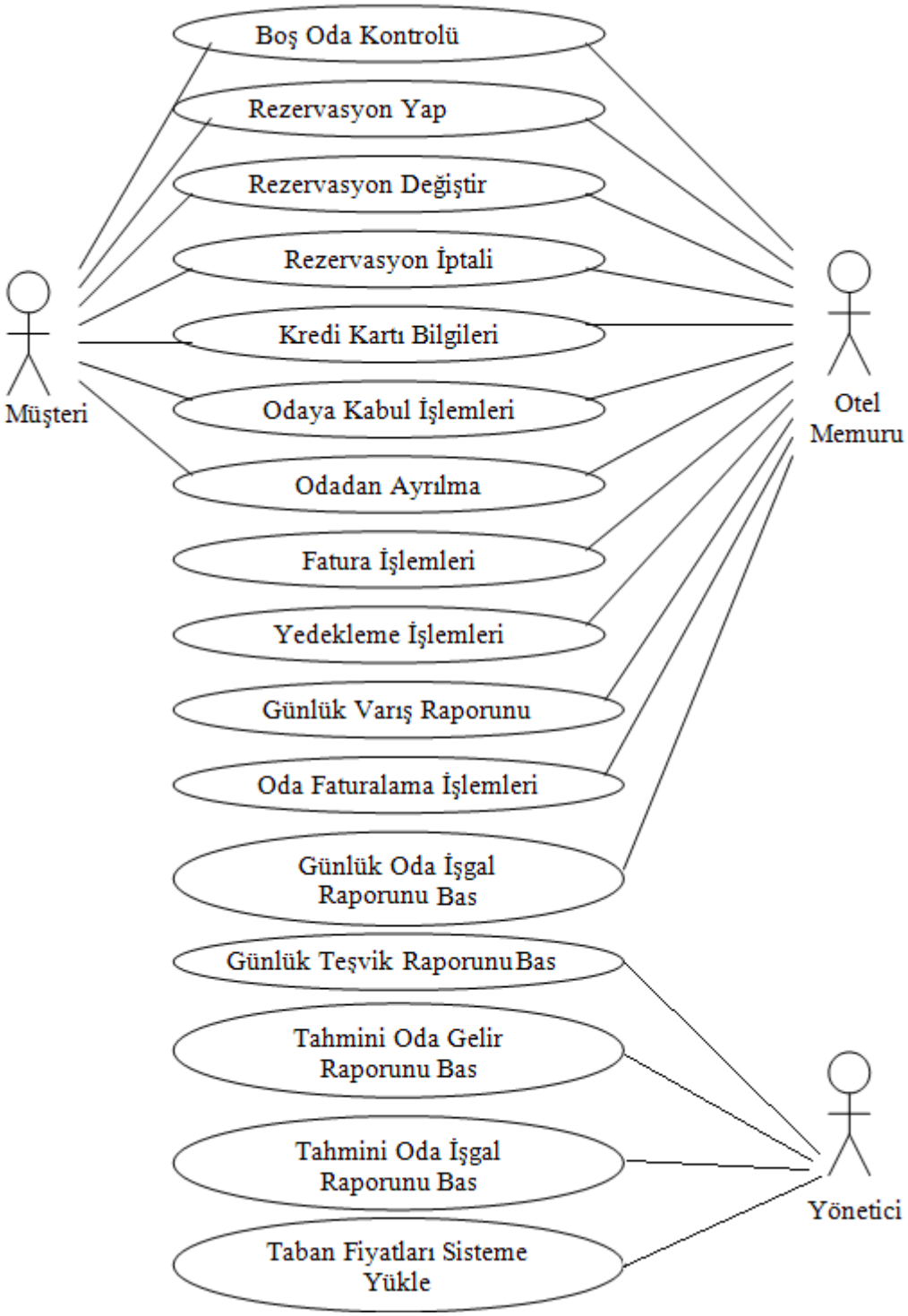
Bu projede otel otomasyonu ile ilgili rezervasyon ve faturalandırma işlemleri yapılmaktadır. Projenin genel kullanım senaryoları şeması izleyen sayfada Şekil 4.1'de verilmiştir.

Bu tez kapsamında 17 adet test yapılmaktadır. Test edilen sınıflar şu şekildedir:

- Get_Input – 4 adet test
- Employee – 8 adet test
- Employee_File – 3 adet test
- Get_Reservation_Input – 2 adet test

Ophelia' Oasis Otel Otomasyonu projesinde *Employee* nesnesi en aktif rol alan nesnelerden biridir. Rezervasyonlar *Employee* tarafından gerçekleştirilmektedir. Bu nedenle en dikkatle test edilmesi gerekli nesnelerden biridir.

4.1. Oasis Projesi Genel Kullanım Senaryoları Şeması



Şekil 4.1 - Oasis Projesi Genel Kullanım Senaryoları

Kod incelendiğinde *Employee_File* sınıfının *Employee* sınıfını kullandığı görülmektedir. Sınıflar arası etkileşimi daha ayrıntılı görebilmek için *Doxygen* dokümantasyon aracından yararlanılmıştır. Bu dokümanlar Doxygen Dokümanları

olarak ektedir. *Employee* sınıfı hem *Employee_File* hem de *Get_Input* sınıflarına referans etmektedir. Ancak hemen her sınıf da *Get_Input* sınıfına referans etmektedir. Bu nedenle başlangıç noktası olarak *Get_Input* sınıfı seçilmiştir. Aşağıda sınıflar arasındaki etkileşim bir matris ile gösterilmiştir. Referans eden metot solda, referans ettiği sınıf ve kaynak koddaki yeri ise sağ tarafta belirtilmiştir.

	Referanslar		
	Employee	**Employee_File**	**Get_Input**
Employee			
setData()			split() *Employee.cpp, 123. satı*
setInformation()			readNumber() *Employee.cpp, 69. satır*
			readString() *Employee.cpp, 74. satır*
user_search()		findUserID() *Employee.cpp, 90. satır*	
Employee_File			
findUserID	getUserID() *Employe_File.cpp, 29. satır*		
	setData() *Employe_File.cpp, 28. satır*		

Çizelge 4.2

Get_Input sınıfının 15 adet test edilebilir metodu vardır. Ancak zaman kısıtı nedeniyle tüm metotları test etmek yerine *Employee* sınıfının kullandıkları test edilmiştir. Bu metotlar:

- readNumber()
- readString()
- split()

Bunlardan readNumber ve readString metodu klavyeden değer (girdi) beklemektedir. Bu metotları ve diğer klavyeden girdi bekleyen metotları test etmek için şu şekilde bir yöntem kullanılmıştır. Öncelikle ekrana girilecek değerin ne olduğu ile ilgili bir mesaj yazdırılır. Örneğin:

Please enter a maximum of 10 digits string

Daha sonra ise bir alt satırda programın hangi değerle karşılaştırarak test yapacağı yazılır. Örneğin:

Note: Result will be compared with "dummypass1"

Klavyeden girilmesi beklenen değer “dummypass1”. Eğer bundan başka bir değer girilirse test başarısız olmalıdır. Eğer doğru girilirse de test başarılı olmalıdır. Ancak örneğin klavyeden girilen değerin sonuna boşluk, yeni satır karakteri ve benzeri olası karakterler eklenseydi, test başarısız olabilirdi. Burada test edilen şey klavyeden girilen değeri kod parçasının bize olduğu gibi döndürüp döndürmediğidir.

İzleyen bölümlerde Doxygen, CPPUnit, testler ve nasıl hazırlanıldığı açıklanmıştır.

4.2. Doxygen Hakkında

Doxygen C, C++ ve Java dilleriyle yazılmış programlar için kullanılabilen otomasyona sahip bir kod doküman sistemidir. Doxygen dokümanları HTML, RTF, Latex ve Man Pages gibi biçimlerde hazırlayabilir.

Doxygen’i kullanacak her bir proje için bir düzenleme dosyasına ihtiyaç vardır. Bu dosya doxygen.cfg olarak adlandırılır ve genellikle kaynak dosyalarıyla aynı klasörde saklanır. Bu dosyada proje adı parametresine Oasis yazılarak tüm projenin içerilmesi sağlanmıştır.

4.3. CPPUnit Hakkında

CppUnit, Erich Gamma and Kent Beck tarafından geliştirilen JUnit test çerçevesinin C++ bağlantısıdır. CppUnit’in temel amacı C++ programlarının birim testlerinde programcıları desteklemektir.

CppUnit çerçevesindeki sınıflar CPPUNIT_NS makrosu tarafından tanımlanmış bir yerde saklanırlar. Test sınıfı (burada TestClass olarak adlandırıldı) bu yerdeki TestFixture sınıfından türetilir. TestFixture bir dizi test tarafından gerekli olan temel

koşulları sağlamak için kullanılan bir sarıcı (wrapper) sınıftır. Böylece tek bir test tarafından gerekli olacak nesne örnekleri yaratılabilir ve ilk değer atanabilir.

TestClass tanımlandıktan sonra uygulamaya geçilmelidir. CPPUnit makrosu CPPUNIT_TEST_SUITE_REGISTRATION**,** TestClass içinde tanımlı olan test setinin "registry" olarak adlandırılan evrensel test listesine eklenmesini sağlar.

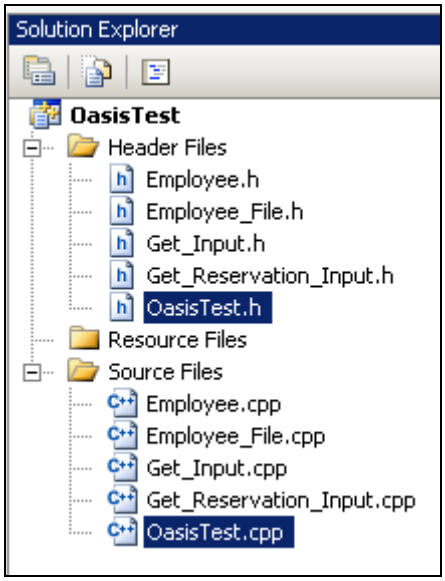
Testlerde CPPUNIT_ASSERT_EQUAL ve CPPUNIT_ASSERT makroları kullanılmıştır. Bunlardan CPPUNIT_ASSERT_EQUAL ilk girilen parametrenin ikinci ile aynı olup olmadığını test eder. Test sonuçlarının listelendiği bölümde ilk parametre *Beklenen Değer* olarak verilmiştir, *Test edilen Değer* ise ekrandan girilen değerdir. Eğer ki 2. parametredeki metot bu değeri döndürürse sonuç *Doğru*, aksi halde *Yanlış* olacaktır. CPPUNIT_ASSERT makrosunda ise girilen cümlenin doğru değer döndürüp döndürmediğine bakılır.

4.4. Teste Hazırlık

Kodun teste hazırlaması ise aşağıdaki gibi gerçekleşmiştir:

1. Bir sınıf tanımlanıp (TestClass) bu sınıf CppUnit çerçevesinde tanımlı TestFixture sınıfından türetilmiştir. Bu tanım OasisTest.h dosyasında yer alır. Sınıf tanımında CPPUNIT_TEST_SUITE makrosuna TestClass parametre olarak verilir.
2. Testler OasisTest.cpp dosyasına TestClass'ın birer metodu gibi yazılır. CPPUNIT_TEST_SUITE_REGISTRATION makrosuna da TestClass parametre olarak verilir.
3. Yaratılan her bir test programı TestClass'ın birer metodu olarak tanımlanır. CPPUNIT_TEST makrosu kullanılarak test programının adı parametre verilerek "CPPUNIT_TEST(testGetInput1)" gibi her bir koçanın çağrılması gerçekleşir. Bu tanımlar da OasisTest.h dosyasında yer alır.
4. Testlerde yer alan dosyalar da aşağıda Şekil 4.2'de listelendiği gibi projeye eklenir ve herhangi bir C++ projesi gibi çalıştırılır.

Daha ayrıntılı bilgi için *Doxygen Dokümanları* altında OasisTest.h ve OasisTest.cpp dosyaları incelenebilir.



Şekil 4.2 – Oasis Test Çözüm Dosyaları

5. SONUÇLAR ve TARTIŞMA

Kaliteli yazılımlar üretebilmenin hedefi mümkün olduğunca hatadan arındırılmış ve müşteri isterlerine uygun sistemler geliştirmektir. Bu hedefe ulaşmak için yazılımın test edilmesi önemli bir süreçtir. Bu çalışmada nesneye yönelik sistemlerde saydam kutu yöntemlerinden yararlanarak sınıf test etme programları yazılmış ve uygulanmıştır.

Bütünüyle hatadan arındırılmış bir yazılım gerçekçi değildir. Diğer taraftan test işlemi zaman ve paraya mal olmaktadır. Bu nedenle mümkün olduğu kadar çok hata yakalayacak test teknikleri geliştirmek çok önemlidir. Bu çalışmada C++ programlama diliyle yazılmış bir proje üzerinde uygulamalı olarak sınıf test etme programları tasarımlanmış ve aynı zamanda bir test aracı olan *CPPUnit* kullanılarak bu testlerin otomasyonu sağlanmıştır. Günümüzde test otomasyonu sürekli gelişen ve değişen yazılımların daha güvenle kullanılabilmesi için testlerin yapılmasında zamanı daha iyi kullanmaya yardımcı olmaktadır ve bu nedenle de önemlidir.

Bu çalışmada kalıtımla ilgili oluşabilecek hatalar için sınıf testleri geliştirilmiştir. Nesneye yönelik sistemlerde test etmeye sınıf seviyesinden başlanabilmektedir. Sınıf testinde test edilen en küçük birim o sınıftan üretilen nesnedir. Nesnenin de çeşitli metotları vardır ve bu metotların bazıları hiyerarşik bir üst sınıftan kalıtımla alınmış olabilir. Kalıtım nesneye yönelik sistemlerin testlerini oldukça karmaşıklaştıran en önemli kavramlardan biridir. Kalıtımla ilgili oluşabilecek hataları bulabilmek amacıyla sınıf testleri için önerilen iki aşamalı bir yöntem kullanılmıştır. Bu yönteme göre birinci aşama yassılaştırılmış sınıf testidir. Burada bir taban sınıf seçilip, öncelikle bu sınıfın testi gerçekleştirilmektedir. İkinci aşama ise artımlı olarak sınıf testinin gerçekleştirildiği aşamadır.

Yukarıda önerilen yönteme göre testleri yapabilmek için öncelikle sınıflar arası etkileşimi görmek gereklidir. Bu amaçla *Doxygen* dokümantasyon aracından yararlanılmış ve diğer sınıflar tarafından referans edilen bir sınıf taban sınıf olarak seçilmiştir. Sınıflar arası etkileşimi gösterebilmek için, taban sınıfta diğer sınıfların

referans ettiği metotlar kullanılarak teste başlanmıştır. Böylece testlerin ilk aşaması olan yassılaştırılmış sınıf testi yapılmıştır. Daha sonra taban sınıfa referans eden diğer sınıflardan bir kaçı üzerinde testler yapılmıştır. Bu da artımlı sınıf testi uygulamasıdır. Böylece nesneye yönelik sistemlerde saydam kutu yöntemlerini kullanarak en alt seviyede sınıf testinin iki aşamada gerçekleştirilmesi üzerine bir çalışma tamamlanmıştır. Aynı zamanda testlerin otomasyonu gerçekleştirilerek programcının zamandan tasarruf etmesi sağlanmıştır.

6. CPPUNIT TESTLERİ

6.1. Get_Input Sınıfı Testleri – Test1 – Test4 (4 test)

6.1.1. Test 1: testGetInput1

```
void TestClass::testGetInput1()

{

        Get_Input *gi = new Get_Input();

        printf("\nPlease enter a maximum of 4 digits number\nNote: Result will be compared with 1234\n");

        CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Input Class - readNumber test has failed", 1234,  (const int)gi->readNumber(4));

}
```

Test edilen metot: readNumber

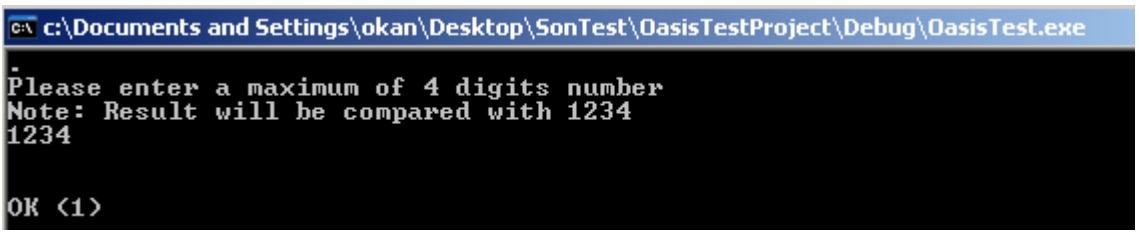
Ait olduğu sınıf: Get_Input

Beklenen değer: 1234

```
// Function to read in the member or provider number
// Input: none
// Output: int containing the number of provider or member entered in by the user
int Get_Input::readNumber(int maxvalue)
{
        int n = 0;
        fflush(stdin);
        scanf("%i", &n);
        //checking to make sure they entered in the correct value of number possible
        if (log10((double)n) > (double) maxvalue)
        {
printf("Incorrect input. Please type an integer of %i digits.\n", maxvalue);
                return readNumber(maxvalue);
        }
        return n;
} // end of readNumber
```

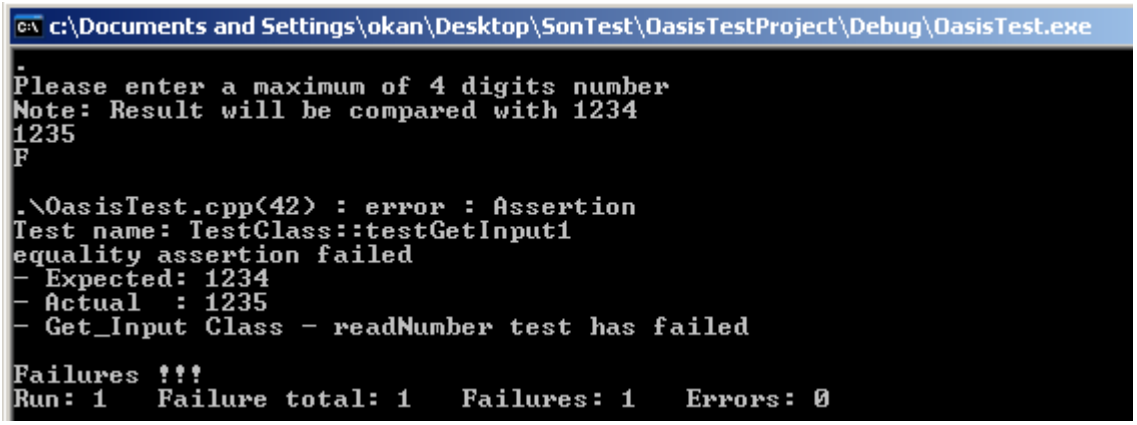
Test edilen değer: 1234

Sonuç: Doğru



Test edilen değer: 1235

Sonuç: Yanlış



6.1.2. Test 2: testGetInput2

```
void TestClass::testGetInput2()
{
        Get_Input *gi = new Get_Input();
printf("\nPlease enter a maximum of 10 digits string\nNote: Result will be
compared with \"dummypass1\"\n");
        CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Input Class - readString
test has failed",
                                                                (const
std::string)"dummypass1",
                                                                (const std::string)gi-
>readString(10));
}
```

Test edilen metot: readString

Ait olduğu sınıf: Get_Input

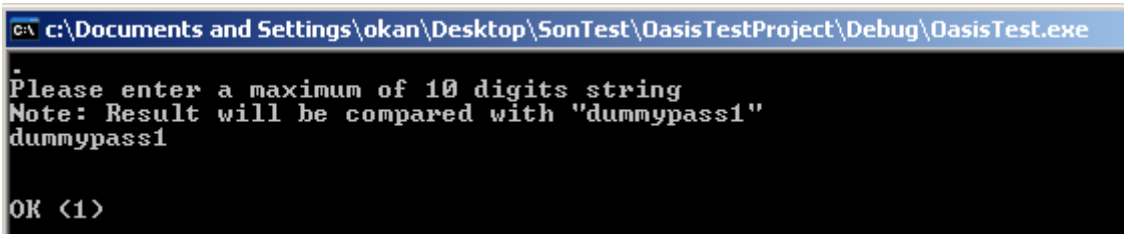
Beklenen değer: dummypass1

```
// Get_Input: interface between dos prompt and user getting information from screen
// Called by: All other classes functions
// Calls: Person.getYear()

char *Get_Input::readString(int maxlength)
{
      char *newString;
                int done = 0;
                int length;
                newString = (char *)calloc (sizeof(char),maxlength);
      while (done != 1)
                {
                        fflush(stdin);
                        gets(newString);
                        length = strlen(newString);
                        if (length >= (maxlength+1))
                        {
                                printf("String entered is too long, please enter a string
with max chars %i\n", maxlength);
                        }
                        else if ((length == 0) && (maxlength != 100))
                        {
                                printf("You must enter a value.  Please try again.\n");
                        }
                        //if string is of type zipcode(length 5) or state(length 2) make
sure they enter exactly that amount of digits
```

```
                        else if ((maxlength == 5 && strlen(newString)!= 5) ||
(maxlength == 2 && strlen(newString) != 2))
                        {
                                printf("String entered is not the right length, please
enter a string with chars %i\n", maxlength);
                        }
                        else done = 1;
                } //end of while !done
                return(newString);
} //end of readString
```

Test edilen değer: dummypass1

Sonuç: Doğru



Test edilen değer: dummypass2

Sonuç: Yanlış

6.1.3. Test 3: testGetInput3

```
void TestClass::testGetInput3()
{
        char **bufferSplit;
        Get_Input *gi = new Get_Input();
        gi->Split("this#is#a#test#sentence", '#', &bufferSplit);

        CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Input Class - Split test has
failed",
                                                        (const std::string)"test",
                                                        (const
std::string)bufferSplit[3]);
}
```

Test edilen metot: Split

Ait olduğu sınıf: Get_Input

Beklenen değer: test

```
// Function split an input string up into a list of other strings based
//   on a split character (which is removed)
// Input: A character string
// Output: Returns the number of substrings created.  The third argument
//                      to the function will contain a char ** to a list of character
//                      strings.
int Get_Input::Split(char *input, char split_char, char ***output)
{
// ** Note:  the output array is really a pointer to a variable-length
//                  array of strings (which themselves are character arrays).

        int i, count, len;
        char buffer[1024];
        count = i = 0;

//First count the total number of split characters - we need to find the
// size of our first dimension for the output
        for ( ; input[i] != '\0'; i++)
                if (input[i] == split_char)
                        count++;

//We preincrement count because we will have a number of strings equal
// to the number of split characters + 1
        *output = (char **)malloc(sizeof(char *) * ++count);
        //*output = new char *[++count];
        count = 0;  len = 0;
```

```
        //Now run through our input again.  Build up a buffer for each
        // substring and when we see a split character, chop the buffer
        // off and copy it into the output array at the proper spot.
        for (i = 0; input[i] != '\0'; i++)
        {
                if (input[i] == split_char)
                {
//The "count" element of our output array needs to have space allocated
                        //(*output)[count] = new char[len+1];
                        (*output)[count] = new char[1024];
                        //We need to add a string terminator manually
                        buffer[len] = '\0';

//And copy the result into the "count" position of the output array
                        strcpy((*output)[count++], buffer);
                        len = 0;
                } else {
//Set the next character in buffer and post-increment our length counter
                        buffer[len++] = input[i];
                }
        }

//We never found a final split character again, so we need to place the
// remaining string in the buffer into the last element of the output array
(*output)[count] = new char[1024];
buffer[len] = '\0';
strcpy((*output)[count++], buffer);
return count;
}
```

Split metodu ilk parametrede verilen karakter katarında ikinci parametrede verilen ayırıcı karakteri bularak parçalara böler ve her bir parçayı 3. parametrede verilen diziye yeni bir eleman olarak atar. Test için kullanılan örnek aşağıdadır:

```
char **bufferSplit;
Get_Input *gi = new Get_Input();
gi->Split("this#is#a#test#sentence", '#', &bufferSplit);
```

Kelimelerin arasına # karakteri koyarak (aynı Employee_File.txt'de olduğu gibi) "this#is#a#test#sentence" ilk parametre olarak verildi. Bu işlemden sonra değerlerin şu şekilde olması beklenir:

bufferSplit[0] = "this"

bufferSplit[1] = "is"

bufferSplit[2] = "a"

bufferSplit[3] = "test"

bufferSplit[4] = "sentence"

Burada bir örnek seçilerek 4. kelimeyi yani dizinin 3 numaralı elemanı test edildi, bu nedenle beklenen değer test kelimesidir.

```
gi->Split("this#is#a#test#sentence", '#', &bufferSplit);
CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Input Class - Split test has failed",
                             (const std::string)"test",
                             (const std::string)bufferSplit[3]);
```

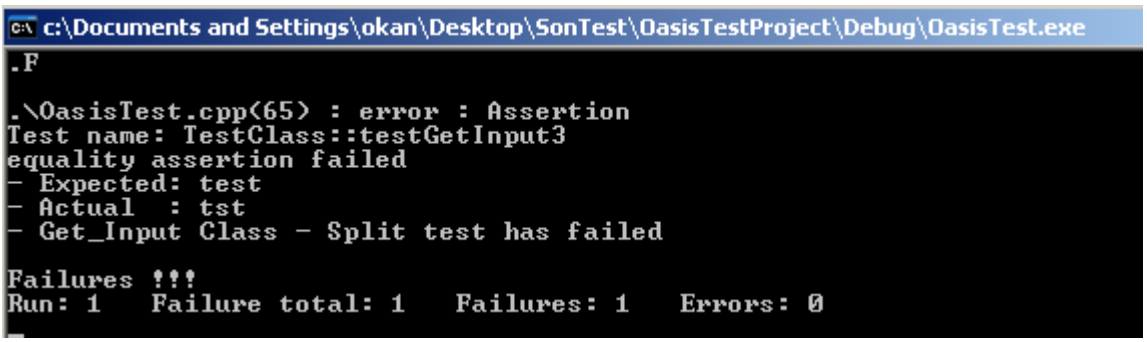
Test edilen değer: test

Sonuç: Doğru

gi->Split("this#is#a#tst#sentence", '#', &bufferSplit);

Test edilen değer: tst

Sonuç: Yanlış



6.1.4. Test 4: testGetInput4

```
// TestClass method: testGetInput4
void TestClass::testGetInput4()
{
	char **bufferSplit;
	int selecteditem;
	char* selectedtext;

	Get_Input *gi = new Get_Input();
	gi->Split("this#is#a#test#sentence", '#', &bufferSplit);
	printf("\nPlease enter a number between 0-3\n");

	selecteditem=(const int)gi->readNumber(1);
	if (selecteditem <4 || selecteditem==0)
	{
		selectedtext=bufferSplit[selecteditem];
		CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Input Class - Split
test has failed",
(const
std::string)selectedtext,
(const
std::string)bufferSplit[selecteditem]);
	}
	else
		printf("You entered a number out of range");
}
```

Bu testte bir öncekine yakın bir kod kullanılmış, kullanıcıdan test edilen eleman indisini ekrandan girmesi istenmiştir.

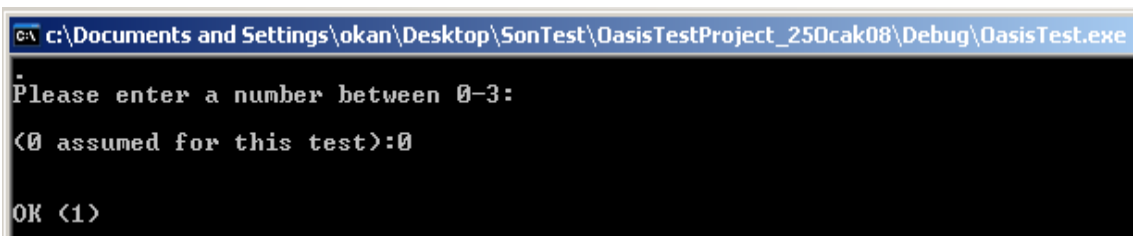
Test edilen metot: Split

Ait olduğu sınıf: Get_Input

Beklenen değer: this (ekranda 0 girildiği var sayıldı)

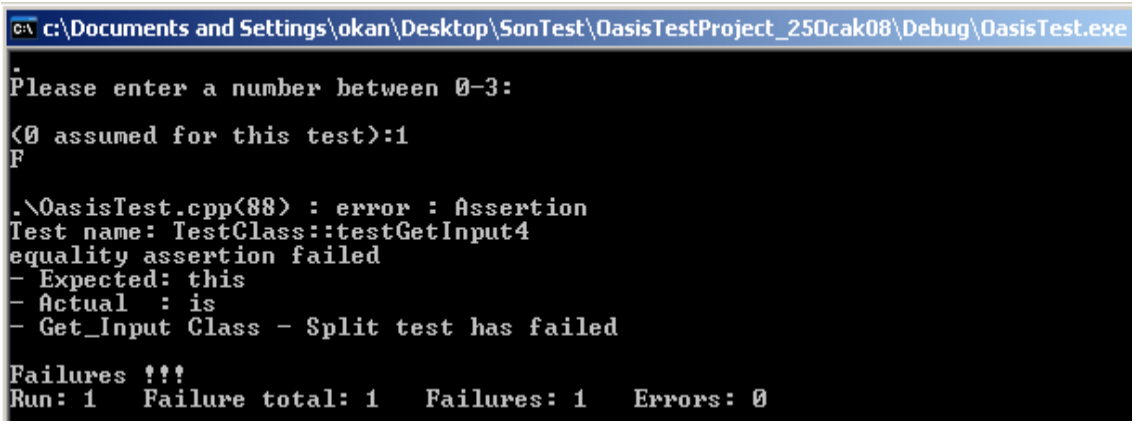
Test edilen değer: 0

Sonuç: Doğru



Test edilen değer: 1

Sonuç: Yanlış



6.2. Employee Sınıfı Testleri – Test 5 - Test 12 (8 test)

6.2.1. Test 5: testEmployee1

```
void TestClass::testEmployee1()
{
	Employee *e = new Employee();
	e->setUserID(15);

	CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - UserID test has failed",15,(const int)e->getUserID());
}
```

Test edilen metot: **getUserID**, setUserID

Ait olduğu sınıf: Employee

Beklenen değer: 15

```
void Employee::setUserID(long tempUserID)
{
	userID = tempUserID;
}

long Employee::getUserID()
{
	return userID;
}
```

Test edilen değer: 15

Sonuç: Doğru

e->setUserID(**14**);
CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - UserID test has failed", **15**, (const int)e->getUserID());

Test edilen değer: 14

Sonuç: Yanlış

6.2.2. Test 6: testEmployee2

```
void TestClass::testEmployee2()
{
	Employee *e = new Employee();
	e->setUserName("Ahmet");

	CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - UserName
test has failed",
                                                    (const
std::string)"Ahmet",
                                                    (const std::string)e-
>getUserName());

	/*                                              (const std::string)"
Ahmet",25 karakterde saga yasliyor*/
}
```

Test edilen metot: **getUserName**, setUserName

Ait olduğu sınıf: Employee

Beklenen değer: Ahmet

```
void Employee::setUserName(char *tempUserName)
{
	sprintf(userName, "%25s", tempUserName);
}

char *Employee::getUserName(void)
{
	return userName;
}
```

Test edilen değer: Ahmet

Sonuç: Yanlış

```
c:\Documents and Settings\okan\Desktop\SonTest\OasisTestProject\Debug\OasisTest.exe
.F

.\OasisTest.cpp(88) : error : Assertion
Test name: TestClass::testEmployee2
equality assertion failed
- Expected: Ahmet
- Actual  :                         Ahmet
- Employee Class - UserName test has failed

Failures !!!
Run: 1   Failure total: 1   Failures: 1   Errors: 0
```

userName'in değeri baş kısmına boşluk doldurulmuş, Ahmet adı sağa yaslanmış toplam 25 karakterlik bir alanda. Projenin bu kısmı, setUserName metodu hatalıdır.

6.2.3. Test 7: testEmployee3

```
void TestClass::testEmployee3()
{
	Employee *e = new Employee();
	e->setUserPassword("qwe123");

	CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - 
UserPassword test has failed",
                                                    (const 
std::string)"qwe123",
                                                    (const std::string)e-
>getUserPassword());

	/*                                              (const std::string)"
qwe123",25 karakterde saga yasliyor*/
}
```

Test edilen metot: **getUserPassword**, setUserPassword

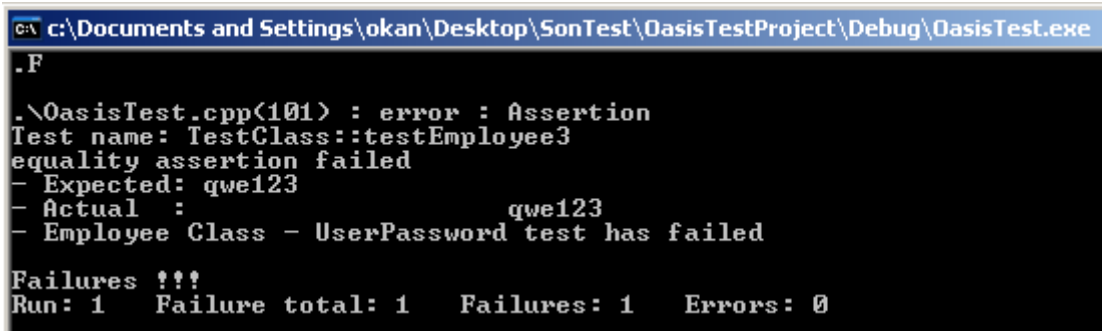
Ait olduğu sınıf: Employee

Beklenen değer: qwe123

```
void Employee::setUserPassword(char *tempUserPassword)
{
	sprintf(userPassword, "%25s", tempUserPassword);
}

char *Employee::getUserPassword(void)
{
	return userPassword;
}
```

Test edilen değer: qwe123

Sonuç: Yanlış



userPassword'ün değeri baş kısmına boşluk doldurulmuş, qwe123 değeri sağa yaslanmış toplam 25 karakterlik bir alanda. Projenin bu kısmı, setUserPassword metodu hatalıdır.

6.2.4. Test 8: testEmployee4

```
void TestClass::testEmployee4()
{
        Employee *e = new Employee();
        e->setUserType(2);

        CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - UserType
test has failed",
                                                        2,
                                                        (const int)e-
>getUserType());
}
```

Test edilen metot: **getUserType**, setUserType

Ait olduğu sınıf: Employee

Beklenen değer: 2

```
void Employee::setUserType(int tempUserType)
{
        userType = tempUserType;
}

int Employee::getUserType(void)
{
        return userType;
}
```

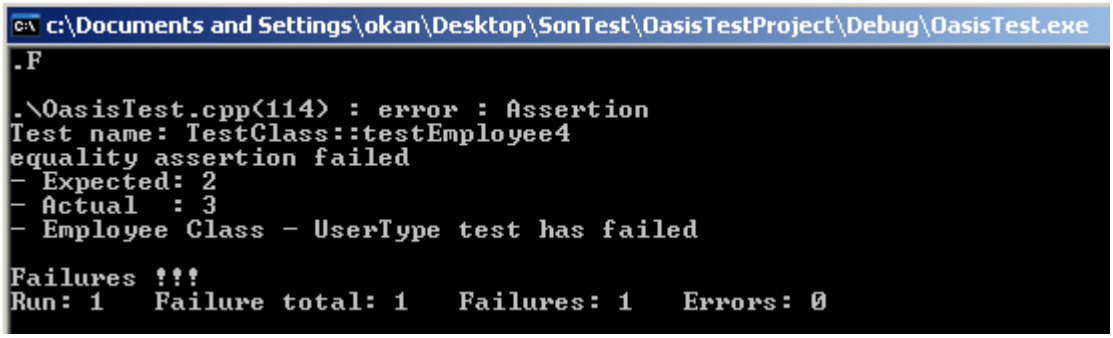
Test edilen değer: 2

Sonuç: Doğru

```
e->setUserType(3);
CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - UserType test has
failed", 2, (const int)e->getUserType());
```

Test edilen değer: 3

Sonuç: Yanlış



6.2.5. Test 9: testEmployee5

```
void TestClass::testEmployee5()
{
        Employee *e = new Employee();
        char *fileData = "1234561#password1#1#Employee1\n";
        e->setData(fileData);

        CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData &
getUserID test has failed",
                                                        1234561,
                                                        (const int)e-
>getUserID());
}
```

Test edilen metot: **getUserID, setData**, setUserID

Ait olduğu sınıf: Employee

Beklenen değer: 1234561

```
// Function setting generic data gotten from file to class vars
// Input: buffer string with data for employee to set
// Output: none
void Employee::setData(char *buffer)
{
                char **bufferSplit;

                Get_Input newInput;
                newInput.Split(buffer, '#', &bufferSplit);

                setUserID(atol(bufferSplit[0]));
                setUserPassword(bufferSplit[1]);
                setUserType(atoi(bufferSplit[2]));
                setUserName(bufferSplit[3]);
} //end of setData

void Employee::setUserID(long tempUserID)
{
        userID = tempUserID;
}
long Employee::getUserID()
{
        return userID;
}
```

Bu testte öncelikle fileData isimli bir karakter katarı tanımlanıyor. Bu değer Employee_File.txt dosyasının içeriğini temsil ediyor. Sonunda yeni satır karakteri (\n) bulunan ve # karakterleriyle ayrılmış bir karakter katarı. Get_Input'un Split metodu daha önce test edilmişti. setData metoduna bu girdi verildiğinde Split metodu veriyi parçaladıktan sonra oluşan dizinin ilk parçasını userID, ikinci parçasını userPassword, üçüncü parçasını userType, dördüncü ve son parçasını ise userName olarak Employee sınıfına ait nesneye yerleştiriyor. Test sırasında bu işlemden sonra getUserID metodundan dönen sonucun beklenen değere uyup uymadığı kontrol ediliyor.

Burada iki sınıfın etkileşimi gerçekleşiyor. Employee sınıfının setData metodu Get_Input sınıfının Split metodunu çağırıyor. Bu metod daha önce kendi sınıfı içinde başarılı bir şekilde test edildiğinden bir başka sınıf referans ettiğinde de testinin başarılı olması beklenmektedir.

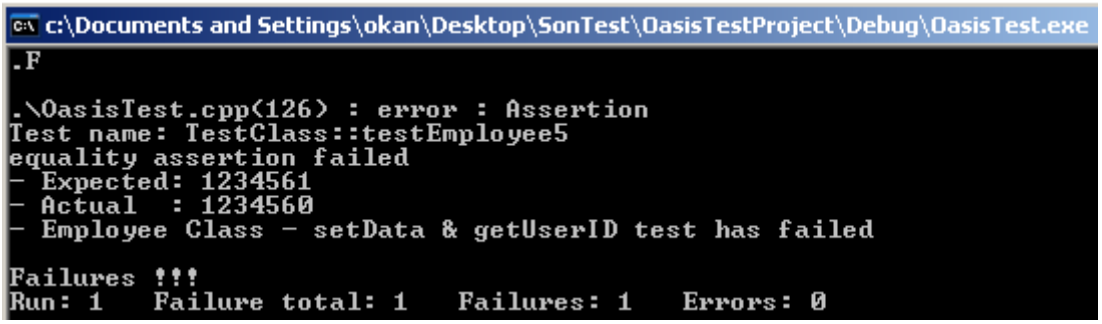
Test edilen değer: 1234561

Sonuç: Doğru

Test edilen değer: 1234560

Sonuç: Yanlış

```
char *fileData = "1234560#password1#1#Employee1\n";
e->setData(fileData);
CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData & getUserID
test has failed",1234561,(const int)e->getUserID());
```



6.2.6. Test 10: testEmployee6

```
void TestClass::testEmployee6()
{
        Employee *e = new Employee();
        char *fileData = "1234561#password1#1#Employee1\n";
        e->setData(fileData);

        CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData &
getUserPassword test has failed",
                                                        (const
std::string)"password1",
                                                        (const std::string)e-
>getUserPassword());
}
```

Test edilen metot: **getUserPassword, setData**, setUserPassword

Ait olduğu sınıf: Employee

Beklenen değer: password1

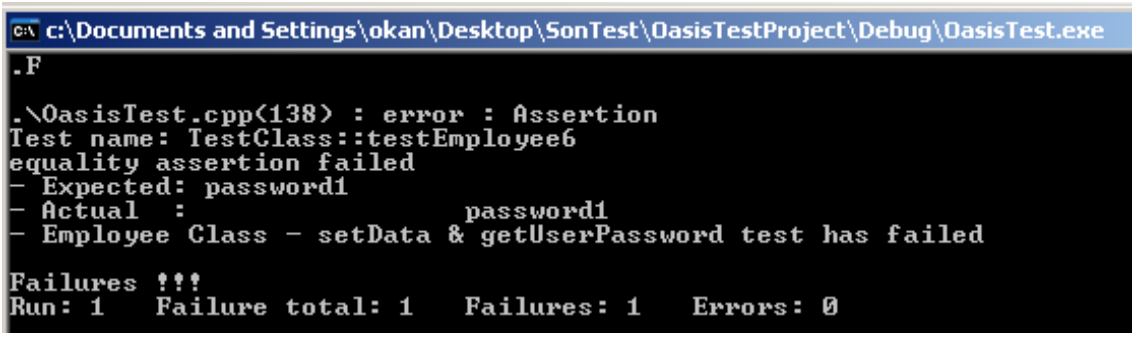
```
// Function setting generic data gotten from file to class vars
// Input: buffer string with data for employee to set
// Output: none
void Employee::setData(char *buffer)
{
                char **bufferSplit;
                Get_Input newInput;
                newInput.Split(buffer, '#', &bufferSplit);

                setUserID(atol(bufferSplit[0]));
                setUserPassword(bufferSplit[1]);
                setUserType(atoi(bufferSplit[2]));
                setUserName(bufferSplit[3]);
} //end of setData
void Employee::setUserPassword(char *tempUserPassword)
{
    sprintf(userPassword, "%25s", tempUserPassword);
}
char *Employee::getUserPassword(void)
{       return userPassword;
}
```

Bu test ve iki sonraki test 8. teste çok benziyor. Testlerin tek bir fonksiyon içersinde yapılmamasının nedeni testlerden biri başarısız olursa aynı fonksiyonda yer alan diğer testlerin yapılmaması. Yani eğer 4 testi birden tek fonksiyonda yapsaydık,

testlerden diyelim ki ikincisi hata verirse üçüncü ve dördüncü test hiç denenmeden atlanırdı. Bunun için her bir fonksiyon içerisinde yalnızca bir test yapıyoruz. O test başarısız olsa bile diğer testler etkilenmiyor.

Test edilen değer: password1

Sonuç: Yanlış



userPassword nesnesinin değeri baş kısmına boşluk doldurulmuş, "password1" değeri sağa yaslanmış toplam 25 karakterlik bir alanda. Daha önce bu kısımda hata bulunmuştu (Test 6), o nedenle bu testi de olumsuz etkilemiştir.

6.2.7. Test 11: testEmployee7

```
void TestClass::testEmployee7()
{
	Employee *e = new Employee();
	char *fileData = "1234561#password1#1#Employee1\n";
	e->setData(fileData);

	CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData &
getUserType test has failed",
                                                        1,
                                                        (const int)e-
>getUserType());
}
```

Test edilen metot: **getUserType, setData**, setUserType

Ait olduğu sınıf: Employee

Beklenen değer: 1

```
// Function setting generic data gotten from file to class vars
// Input: buffer string with data for employee to set
// Output: none
void Employee::setData(char *buffer)
{
		char **bufferSplit;

		Get_Input newInput;
		newInput.Split(buffer, '#', &bufferSplit);

		setUserID(atol(bufferSplit[0]));
		setUserPassword(bufferSplit[1]);
		setUserType(atoi(bufferSplit[2]));
		setUserName(bufferSplit[3]);
} //end of setData
void Employee::setUserType(int tempUserType)
{
	userType = tempUserType;
}

int Employee::getUserType(void)
{   return userType;
}
```

Test edilen değer: 1

Sonuç: Doğru

Test edilen değer: 2

Sonuç: Yanlış

char *fileData = "1234561#password1#**2**#Employee1\n";
e->setData(fileData);

CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData & getUserType test has failed", 1, (const int)e->getUserType());

6.2.8. Test 12: testEmployee8

```
void TestClass::testEmployee8()
{
	Employee *e = new Employee();
	char *fileData = "1234561#password1#1#Employee1\n";
	e->setData(fileData);

	CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData &
getUserName test has failed",
                                                    (const
std::string)"Employee1",
                                                    (const std::string)e-
>getUserName());
}
```

Test edilen metot: **getUserName, setData**, setUserName

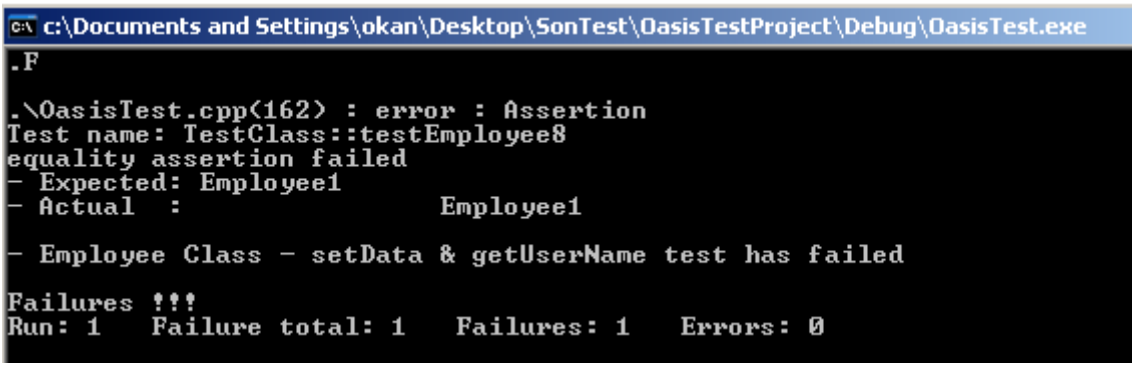
Ait olduğu sınıf: Employee

Beklenen değer: Employee1

```
// Function setting generic data gotten from file to class vars
// Input: buffer string with data for employee to set
// Output: none
void Employee::setData(char *buffer)
{
		char **bufferSplit;

		Get_Input newInput;
		newInput.Split(buffer, '#', &bufferSplit);

		setUserID(atol(bufferSplit[0]));
		setUserPassword(bufferSplit[1]);
		setUserType(atoi(bufferSplit[2]));
		setUserName(bufferSplit[3]);
} //end of setData
void Employee::setUserName(char *tempUserName)
{
	sprintf(userName, "%25s", tempUserName);
}

char *Employee::getUserName(void)
{
	return userName;
}
```

Test edilen değer: Employee1

Sonuç: Yanlış



userName nesnesinin değeri baş kısmına boşluk doldurulmuş, "Employee1" değeri sağa yaslanmış toplam 25 karakterlik bir alanda. Daha önce bu kısımda hata bulunmuştu (Test 5), o nedenle bu testi de olumsuz etkilemiştir.

6.3. Employee_File Sınıfı Testleri – Test13 - Test15 (3 test)

6.3.1. Test 13: testEmployeeFile1

```
void TestClass::testEmployeeFile1()
{
        Employee_File *ef = new Employee_File();
        std::string user;

        user = ef->findUserID(1234561);
        std::cout << user;
        CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee_File Class -
FindUserID test has failed",
                                                        (const
std::string)"1234561#password1#1#Employee1\n",
                                                        (const std::string)user);
}
```

Test edilen metot: findUserID

Ait olduğu sınıf: Employee_File

Beklenen değer: 1234561#password1#1#Employee1

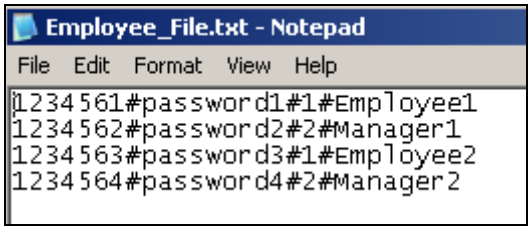
```
char *Employee_File::findUserID(int searchNumber)
{
        char *buffer;
        buffer = (char *)malloc(1024);

        Employee foundEmployee;
        try
   {
                FILE *file;
                if ((file = fopen("Employee_File.txt", "r")) == NULL)
                {
                        printf("Cannot open Employee_File.txt file\n");
                        exit(1);
                }
                else
                {
                  //finding the name in the file if it exisits
  while (!feof(file))
                    {
                        fgets(buffer, 1024, file);

                        foundEmployee.setData(buffer);
```

```
                        if (foundEmployee.getUserID() == searchNumber)
                                return(buffer);

                    } //end of while
                }  //end of else
        }
        catch (int i) {printf("Reading file IO Exception: %i", i);}

        printf("Number entered is not found in file. Please try again.\n\n");
        return (0);
}
```

```
void Employee::setData(char *buffer)
{
                char **bufferSplit;

                Get_Input newInput;
                newInput.Split(buffer, '#', &bufferSplit);

                setUserID(atol(bufferSplit[0]));
                setUserPassword(bufferSplit[1]);
                setUserType(atoi(bufferSplit[2]));
                setUserName(bufferSplit[3]);

} //end of setData
```

```
long Employee::getUserID()
{
        return userID;
}
```

Employee_File sınıfı tek bir metoddan ibarettir. Bu method, Employee_File.txt ismindeki dosyanın içersinde satır satır ilerleyerek parametreden vermiş olduğumuz userID'ye ait bilgileri döndürmeye yarıyor. Bu bilgileri Employee_File.txt dosyasının userID'yi bulduğu satırını olduğu gibi (yeni satır karakteri de (\n) dahil) text formatında döndürerek alıyor. İlk iki testte (12 ve 13) dosyanın birinci ve ikinci satırlarındaki userID'ler kullanılmıştır. Dosyanın ilk satırı da sonraki satırı da başarılı bir şekilde bulunabilmiştir.



Test edilen değer: 1234561

Sonuç: Doğru

6.3.2. Test 14: testEmployeeFile2

```
void TestClass::testEmployeeFile2()
{
        Employee_File *ef = new Employee_File();
        std::string user;

        user = ef->findUserID(1234562);
    std::cout << user;
        CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee_File Class -
FindUserID test has failed",
                                                        (const
std::string)"1234562#password2#2#Manager1\n",
                                                        (const std::string)user);
}
```

Test edilen metot: findUserID

Ait olduğu sınıf: Employee_File

Beklenen değer: 1234562#password2#2#Manager1

Test edilen değer: 1234562

Sonuç: Doğru

6.3.3. Test 15: testEmployeeFile3

```
void TestClass::testEmployeeFile3()
{
        Employee_File *ef = new Employee_File();
        char *user;

        user = ef->findUserID(1234565);
        CPPUNIT_ASSERT_MESSAGE("Employee_File Class - User doesn't
exist",
                                                user != 0);
}
```

Test edilen metot: findUserID

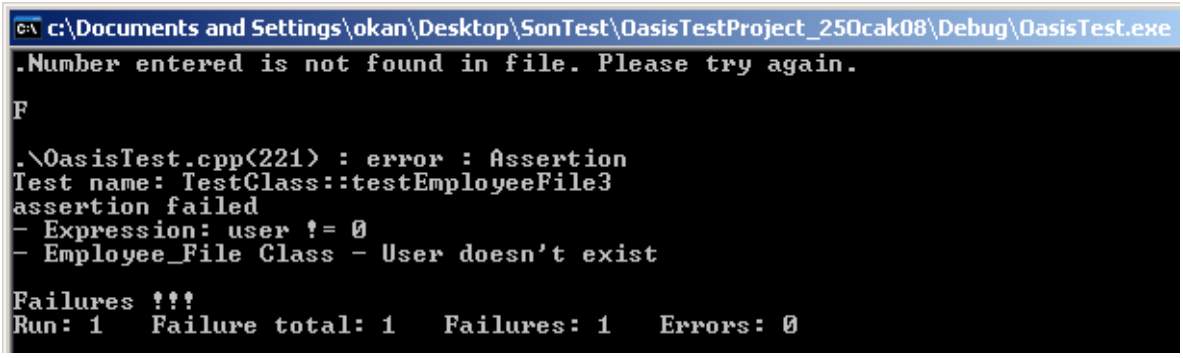
Ait olduğu sınıf: Employee_File

Beklenen değer: 0

Bu testte dosyada yer almayan bir userID kullanılmıştır. Employee_File sınıfının findUserID metodu eğer userID'yi dosyada bulamazsa 0 (sıfır) döndürüyor. Bu durumda kullanıcın varolması için koşul user != 0 olmalıdır. Bu kez CPPUNIT_ASSERT_EQUAL_MESSAGE kullanmak yerine CPPUNIT_ASSERT_MESSAGE kullanarak sonucun belirli bir koşula uyup uymadığı gözlenmiştir.

Test edilen değer: 1234565

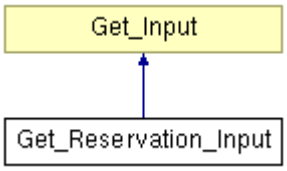
Sonuç: 0



6.4. Get_Reservation_Input Sınıfı Testleri – Test16 - Test17 (2 test)

Ön Bilgi

Inheritance diagram for Get_Reservation_Input:



Class definition:

```
class Get_Reservation_Input : public Get_Input
{
public:
        char readReservationType(void);
        char *readCreditCardExpiration(void);

};
```

6.4.1. Test 16: testGetReservationInput1

```
void TestClass::testGetReservationInput1()
{
        Get_Reservation_Input *gri = new Get_Reservation_Input();
        printf("\nPlease enter a reservation type (I,D,C or P)\nNote: Result will be
compared with \'D\'\n");

        CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Reservation_Input Class -
readReservationType test has failed",
                                                        'D',
                                                        (const char)gri-
>readReservationType());
}
```

Test edilen metot: readReservationType

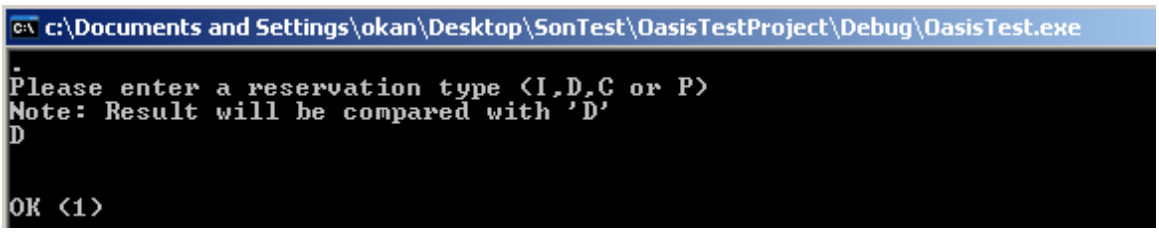
Ait olduğu sınıf: Get_Reservation_Input

Beklenen değer: D

```
//Function: read in reservation selection from user
//Input: none
//Output:  ReservationType user enters
char Get_Reservation_Input::readReservationType(void)
{
	char newType;
	int done = 0;
	int maxlength = 1;

	while (done != 1)
	{
		fflush(stdin);
		scanf("%c", &newType);

		newType = (char)toupper((int)newType);

		switch (newType)
		{
			case 'I':
			case 'D':
			case 'C':
			case 'P':
				done = 1;
				break;
			default:
				printf("Please enter either an I, D, C, or P.\n");
		}

	} //end of while !done

	return (newType);
}
```

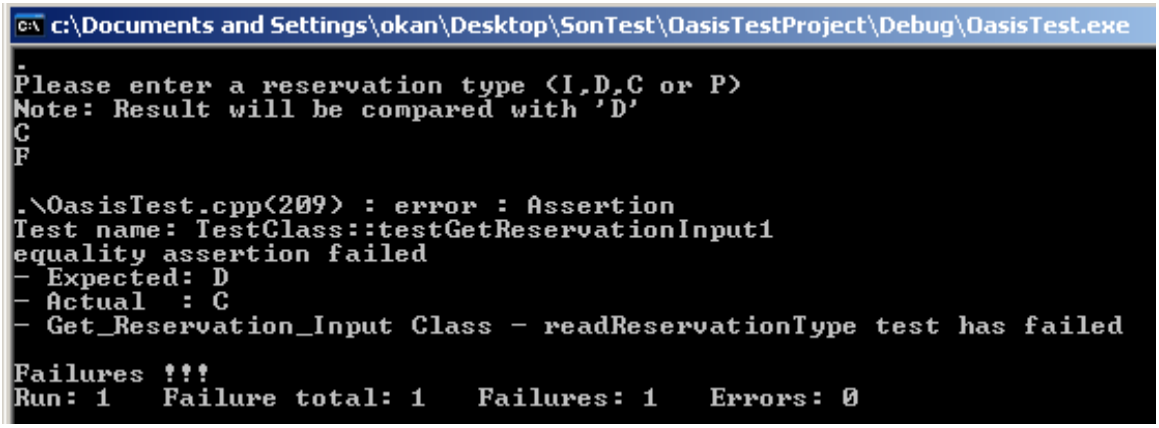
Test edilen değer: D

Sonuç: Doğru



Test edilen değer: C

Sonuç: Yanlış



6.4.2. Test 17: testGetReservationInput2

```
void TestClass::testGetReservationInput2()
{
	Get_Reservation_Input *gri = new Get_Reservation_Input();
	printf("\nPlease enter your credit card expiration date (mm-yyyy)\nNote: Result will be compared with \"07-2012\"\n");

	CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Reservation_Input Class - readReservationType test has failed",
		(const std::string)"07-2012",
		(const std::string)gri->readCreditCardExpiration());
}
```

Test edilen metot: readCreditCardExpiration

Ait olduğu sınıf: Get_Reservation_Input

Beklenen değer: 07-2012

```
// Function to read in the date from user
// Input: maxlength of number of characters for date
// Output: String containing the date
char *Get_Reservation_Input::readCreditCardExpiration(void)
{
	char *tempString;
	unsigned int maxLength = 7;

	struct tm *newtime;
	time_t aclock;
	time( &aclock );
	newtime = localtime( &aclock );
	int currentYear = newtime->tm_year + 1900;

	int done = 0;

	tempString = (char *)calloc(1024, sizeof(char));


	while (done != 1)
	{
		fflush(stdin);
		scanf("%s", tempString);

		if (strlen(tempString) != maxLength)
```

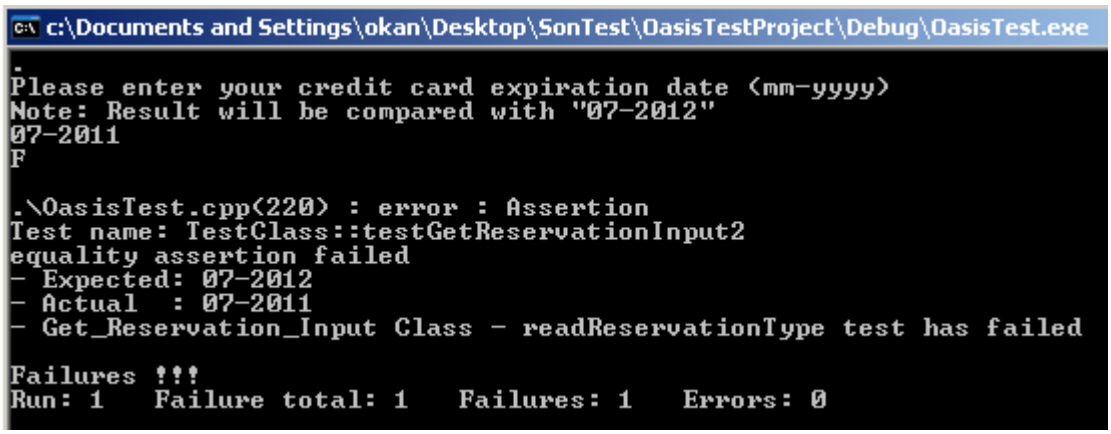
```
		{
			printf("Date entered is not right length, please enter a date
(mm-yyyy) with max chars %u\n", maxLength);
		} else {

			char tempMonth[2] = {tempString[0], tempString[1]};
			char tempYear[4] = {tempString[3], tempString[4],
tempString[5], tempString[6]};

			int month = atoi(tempMonth);
			int year = atoi(tempYear);

			if (month < 1 || month > 12) {
				printf("Month entered in not valid.  Please enter a date
with month between 01 and 12.\n");
			}
			else if (year == 0 || year < currentYear-1)
			{
				printf("Year entered is not valid.  Please enter a date
with year greater than or equal to %i\n", currentYear);
			}
			else
			{
				done = 1;
			}
		}

	} //end of while

	return tempString;
}
```

Test edilen değer: 07-2012

Sonuç: Doğru

Test edilen değer: 07-2011

Sonuç: Yanlış



7. DOXYGEN DÖKÜMANLARI

Bu bölümdeki dokümanlar Doxygen yazılımı tarafından yaratılmıştır.

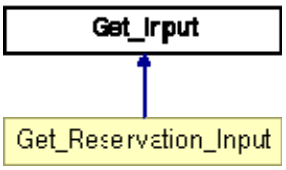
7.1. Oasis Class Hierarchy

- Employee
- Employee_File
- Get_Input
  - Get_Reservation_Input
- Maintain_Daily_Reports
- Maintain_Manager_Reports
- Maintain_Rates
- Maintain_Reservation
  - Maintain_60Day_Reservation
  - Maintain_Conventional_Reservation
  - Maintain_Prepaid_Reservation
- Oasis
- Print_Accomodation_Bill
- Print_Daily_Arrivals_Report
- Print_Daily_Occupancy_Report
- Print_Manager_Expected_Income_Report
- Print_Manager_Expected_Occupancy_Report
- Print_Manager_Incentive_Report
- Rate_File
- Rates
- Reservation
- Reservation_File
- TestClass
- Verify_Employee

7.2. SINIF REFERANSLARI

7.2.1. Get_Input Class Reference

#include <**Get_Input.h**>
Inheritance diagram for Get_Input:



Public Member Functions

char *	**readString** (int)
char *	**readType** (int)
char *	**readDate** (int)
int	**readSelection** (int)
int	**readNumber** (int)
int	**readCode** ()
char	**readStatus** (char, int)
float	**readFloat** ()
long	**readLong** ()
char *	**getCurrentDate** ()
char *	**getNextDate** (char *currentDate)
char *	**getPreviousDate** (char *currentDate)
char *	**getDateInFuture** (char *currentDate, int days)
int	**Split** (char *input, char split_char, char ***output)
char	**readYesNo** ()

Private Member Functions

char *	**addDate** (char *Date, long seconds)

Detailed Description

Definition at line **5** of file **Get_Input.h**.

Member Function Documentation

Bu kısımda yalnızca test edilen metodlar listelenmiştir.

char * Get_Input::readString (int *maxlength*)

Definition at line **16** of file **Get_Input.cpp**.

Referenced by **Reservation::addCreditCardData()**, **Reservation::addGuestEmailAddress()**, **Reservation::getReservationData()**, **Reservation::getUpdateReservationData()**, **Employee::setInformation()**, **TestClass::testGetInput2()**, **Reservation::updateCreditCardData()**, and **Reservation::updateGuestEmailAddress()**.

```
00017 {
00018       char *newString;
00019            int done = 0;
00020            int length;
00021            newString = (char *)calloc (sizeof(char),maxlength);
00022
00023       while (done != 1)
00024            {
00025                 fflush(stdin);
00026                 gets(newString);
00027
00028                 length = strlen(newString);
00029
00030                 if (length >= (maxlength+1))
00031                 {
00032                      printf("String entered is too long, please enter a
string with max chars %i\n", maxlength);
00033                 }
00034                 else if ((length == 0) && (maxlength != 100))
00035                 {
00036                      printf("You must enter a value.  Please try
again.\n");
00037                 }
00038                 //if string is of type zipcode(length 5) or state(length 2)
make sure they enter exactly that amount of digits
00039                 else if ((maxlength == 5 && strlen(newString)!= 5) ||
(maxlength == 2 && strlen(newString) != 2))
```

```
00040                   {
00041                           printf("String entered is not the right length,
please enter a string with chars %i\n", maxlength);
00042                   }
00043                   else done = 1;
00044
00045           } //end of while !done
00046
00047           return(newString);
00048 } //end of readString
```

int Get_Input::readNumber (int *maxvalue*)

Definition at line **194** of file **Get_Input.cpp**.

Referenced by **Maintain_Reservation::cancelReservation()**, **Maintain_Reservation::checkInGuest()**, **Maintain_Reservation::checkOutGuest()**, **Reservation::getReservationData()**, **Reservation::getUpdateReservationData()**, **main()**, **Maintain_Daily_Reports::maintainDailyReportsControl()**, **Maintain_Manager_Reports::maintainManagerReportsControl()**, **Employee::setInformation()**, **TestClass::testGetInput1()**, **TestClass::testGetInput4()**, **Maintain_Prepaid_Reservation::updateReservation()**, **Maintain_Conventional_Reservation::updateReservation()**, and **Maintain_60Day_Reservation::updateReservation()**.

```
00195 {
00196       int n = 0;
00197
00198       fflush(stdin);
00199       scanf("%i", &n);
00200
00201       //checking to make sure they entered in the correct value of
number possible
00202       if (log10((double)n) > (double) maxvalue)
00203       {
00204             printf("Incorrect input. Please type an integer of %i
digits.\n", maxvalue);
00205             return readNumber(maxvalue);
00206       }
00207
```

```
00208       return n;
00209 } // end of readNumber
```

int Get_Input::Split (char * ***input,***
char ***split_char,***
char * *** ***output***
)

Definition at line **399** of file **Get_Input.cpp**.

Referenced by **Reservation_File::addContinuedReservations()**, **Reservation::addRooms()**, **Maintain_Prepaid_Reservation::calculateRoomPrice()**, **Maintain_Conventional_Reservation::calculateRoomPrice()**, **Maintain_60Day_Reservation::calculateRoomPrice()**, **Maintain_Reservation::cancelReservation()**, **Reservation_File::createEmail()**, **Reservation_File::findNumber()**, **Reservation::getAccommodationBill()**, **Reservation::getConfirmation()**, **Reservation_File::getDailyOccupancyInfo()**, **Reservation_File::getDailyRoomsInfo()**, **Reservation::getManagerIncentive()**, **Reservation::getManagerIncome()**, **Reservation::getManagerOccupancy()**, **Reservation_File::getNewNumber()**, **Reservation::getUpdateReservationData()**, **Reservation::noPayment()**, **Reservation::noShowPolicy()**, **Print_Accomodation_Bill::printAccomodationBill()**, **Print_Daily_Occupancy_Report::printDailyOccupancyReport()**, **Print_Daily_Arrivals_Report::printDailyRoomsReport()**, **Print_Manager_Incentive_Report::printManagerIncentiveReport()**, **Print_Manager_Expected_Income_Report::printManagerIncomeReport()**, **Print_Manager_Expected_Occupancy_Report::printManagerOccupancyReport()**, **Reservation::printRoomArrivals()**, **Reservation::printRoomOccupancy()**, **Reservation::reminderEmail()**, **Reservation_File::returnReservations()**, **Maintain_Reservation::roomAvailability()**, **Reservation_File::searchReservations()**, **Reservation_File::searchRoomNumber()**, **Reservation::setData()**, **Employee::setData()**, **TestClass::testGetInput3()**, **TestClass::testGetInput4()**, **Reservation_File::updateReservationFile()**, **Maintain_Prepaid_Reservation::updateRoomPrice()**, and **Maintain_60Day_Reservation::updateRoomPrice()**.

```
00400 {
00401       // ** Note:  the output array is really a pointer to a variable-length
00402       //                   array of strings (which themselves are character
arrays).
00403
00404       int i, count, len;
00405
```

```
00406       char buffer[1024];
00407
00408       count = i = 0;
00409
00410       //First count the total number of split characters - we need to find
the
00411       // size of our first dimension for the output
00412       for ( ; input[i] != '\0'; i++)
00413             if (input[i] == split_char)
00414                   count++;
00415
00416       //We preincrement count because we will have a number of
strings equal
00417       // to the number of split characters + 1
00418       *output = (char **)malloc(sizeof(char *) * ++count);
00419       //*output = new char *[++count];
00420
00421       count = 0;  len = 0;
00422
00423       //Now run through our input again.  Build up a buffer for each
00424       // substring and when we see a split character, chop the buffer
00425       // off and copy it into the output array at the proper spot.
00426       for (i = 0; input[i] != '\0'; i++)
00427       {
00428             if (input[i] == split_char)
00429             {
00430                   //The "count" element of our output array needs to have
space allocated
00431                   //(*output)[count] = new char[len+1];
00432                   (*output)[count] = new char[1024];
00433
00434                   //We need to add a string terminator manually
00435                   buffer[len] = '\0';
00436
00437                   //And copy the result into the "count" position of the
output array
00438                   strcpy((*output)[count++], buffer);
00439                   len = 0;
00440             } else {
```

```
00441                   //Set the next character in buffer and post-increment
our length counter
00442                   buffer[len++] = input[i];
00443               }
00444       }
00445
00446       //We never found a final split character again, so we need to
place the
00447       // remaining string in the buffer into the last element of the output
array
00448       (*output)[count] = new char[1024];
00449       buffer[len] = '\0';
00450       strcpy((*output)[count++], buffer);
00451
00452       return count;
00453 }
```

The documentation for this class was generated from the following files:

- **Get_Input.h**
- **Get_Input.cpp**

7.2.2. Employee Class Reference

#include <**Employee.h**>

Public Member Functions

	Employee ()
void	**setUserID** (long tempUserID)
void	**setUserPassword** (char *tempUserPassword)
void	**setUserType** (int tempUserType)
void	**setUserName** (char *tempUserName)
long	**getUserID** ()
char *	**getUserPassword** ()
int	**getUserType** ()
char *	**getUserName** ()
int	**userSearch** (int searchnumber)
int	**setInformation** ()
void	**setData** (char *)

Private Attributes

int	**userID**
char *	**userPassword**
int	**userType**
char *	**userName**

Detailed Description

Definition at line **3** of file **Employee.h**.

Constructor & Destructor Documentation

Employee::Employee ( )

Definition at line **9** of file **Employee.cpp**.

References **userName**, and **userPassword**.

```
00010 {
00011         userPassword = (char *)calloc(25, sizeof(char));
00012         userName = (char *)calloc(25, sizeof(char));
00013 }
```

Member Function Documentation

void Employee::setUserID (long *tempUserID*)

Definition at line **31** of file **Employee.cpp**.

References **userID**.

Referenced by **setData()**, **setInformation()**, and **TestClass::testEmployee1()**.

```
00032 {
00033         userID = tempUserID;
00034 }
```

void Employee::setUserPassword (char * *tempUserPassword*)

Definition at line **21** of file **Employee.cpp**.

References **userPassword**.

Referenced by **setData()**, **setInformation()**, and **TestClass::testEmployee3()**.

```
00022 {
00023         sprintf(userPassword, "%25s", tempUserPassword);
00024 }
```

void Employee::setUserType (int *tempUserType*)

Definition at line **26** of file **Employee.cpp**.

References **userType**.

Referenced by **setData()**, **TestClass::testEmployee4()**, and **userSearch()**.

```
00027 {
```

```
00028       userType = tempUserType;
00029 }
```

void Employee::setUserName (char * *tempUserName*)

Definition at line **16** of file **Employee.cpp**.

References **userName**.

Referenced by **setData()**, **TestClass::testEmployee2()**, and **userSearch()**.

```
00017 {
00018       sprintf(userName, "%25s", tempUserName);
00019 }
```

long Employee::getUserID ( )

Definition at line **36** of file **Employee.cpp**.

References **userID**.

Referenced by **Employee_File::findUserID()**, **TestClass::testEmployee1()**, **TestClass::testEmployee5()**, **userSearch()**, and **Verify_Employee::verifyAccess()**.

```
00037 {
00038       return userID;
00039 }
```

char * Employee::getUserPassword (void)

Definition at line **41** of file **Employee.cpp**.

References **userPassword**.

Referenced by **TestClass::testEmployee3()**, **TestClass::testEmployee6()**, and **userSearch()**.

```
00042 {
00043       return userPassword;
00044 }
```

int Employee::getUserType (void)

Definition at line **46** of file **Employee.cpp**.

References **userType**.

Referenced by **TestClass::testEmployee4()**, **TestClass::testEmployee7()**, and **userSearch()**.

```
00047 {
00048         return userType;
00049 }
```

char * Employee::getUserName (void)

Definition at line **51** of file **Employee.cpp**.

References **userName**.

Referenced by **TestClass::testEmployee2()**, **TestClass::testEmployee8()**, and **userSearch()**.

```
00052 {
00053         return userName;
00054 }
```

int Employee::userSearch (int *searchnumber*)

Definition at line **82** of file **Employee.cpp**.

References **Employee_File::findUserID()**, **getUserID()**, **getUserName()**, **getUserPassword()**, **getUserType()**, **setData()**, **setUserName()**, and **setUserType()**.

Referenced by **Verify_Employee::verifyAccess()**.

```
00083 {
00084         char *buffer = "\0";
00085
00086         Employee_File newSearch;
00087
00088         Employee foundUser;
00089
00090         buffer = newSearch.findUserID(getUserID());
00091
00092         if (buffer != NULL)
```

```
00093       {
00094             foundUser.setData(buffer);
00095
00096             if (strcmp(foundUser.getUserPassword(),
getUserPassword() ) == 0)
00097             {
00098                   setUserType(foundUser.getUserType());
00099                   setUserName(foundUser.getUserName());
00100
00101                   return(getUserType());
00102             }
00103             else
00104             {
00105                   printf("\nUserID and Password do not match.  Please
try again.\n\n");
00106                   return (0);
00107             }
00108       }
00109       else
00110             return (0);
00111 }
```

int Employee::setInformation ( )

Definition at line **60** of file **Employee.cpp**.

References **Get_Input::readNumber()**, **Get_Input::readString()**, **setUserID()**, and **setUserPassword()**.

Referenced by **Verify_Employee::verifyAccess()**.

```
00061 {
00062       int tempUserID;
00063       char* tempUserPassword ;
00064
00065       Get_Input userInput;
00066
00067
00068       printf("User ID: ");
00069       tempUserID = userInput.readNumber(7);
```

```
00070
00071       setUserID(tempUserID);
00072
00073       printf("User password: ");
00074       tempUserPassword = userInput.readString(10);
00075
00076       setUserPassword(tempUserPassword);
00077
00078       return (1);
00079 } //end of setInformation()
```

void Employee::setData (char * *buffer*)

Definition at line **117** of file **Employee.cpp**.

References **setUserID()**, **setUserName()**, **setUserPassword()**, **setUserType()**, and **Get_Input::Split()**.

Referenced by **Employee_File::findUserID()**, **TestClass::testEmployee5()**, **TestClass::testEmployee6()**, **TestClass::testEmployee7()**, **TestClass::testEmployee8()**, and **userSearch()**.

```
00118 {
00119
00120           char **bufferSplit;
00121
00122           Get_Input newInput;
00123           newInput.Split(buffer, '#', &bufferSplit);
00124
00125           setUserID(atol(bufferSplit[0]));
00126           setUserPassword(bufferSplit[1]);
00127           setUserType(atoi(bufferSplit[2]));
00128           setUserName(bufferSplit[3]);
00129
00130
00131
00132 }
```

Member Data Documentation

int Employee::userID [private]

Definition at line **6** of file **Employee.h**.

Referenced by **getUserID()**, and **setUserID()**.

char* Employee::userPassword [private]

Definition at line **7** of file **Employee.h**.

Referenced by **Employee()**, **getUserPassword()**, and **setUserPassword()**.

int Employee::userType [private]

Definition at line **8** of file **Employee.h**.

Referenced by **getUserType()**, and **setUserType()**.

char* Employee::userName [private]

Definition at line **9** of file **Employee.h**.

Referenced by **Employee()**, **getUserName()**, and **setUserName()**.

The documentation for this class was generated from the following files:

- **Employee.h**
- **Employee.cpp**

7.2.3. Employee_File Class Reference

#include <**Employee_File.h**>

Public Member Functions

char * **findUserID** (int searchNumber)

Detailed Description

Definition at line **3** of file **Employee_File.h**.

Member Function Documentation

char * Employee_File::findUserID (int *searchNumber*)

Definition at line **6** of file **Employee_File.cpp**.

References **Employee::getUserID()**, and **Employee::setData()**.

Referenced by **TestClass::testEmployeeFile1()**, **TestClass::testEmployeeFile2()**, **TestClass::testEmployeeFile3()**, and **Employee::userSearch()**.

```
00007 {
00008       char *buffer;
00009       buffer = (char *)malloc(1024);
00010 
00011       Employee foundEmployee;
00012 
00013       try
00014     {
00015             FILE *file;
00016             if ((file = fopen("Employee_File.txt", "r")) == NULL)
00017             {
00018                   printf("Cannot open Employee_File.txt file\n");
00019                   exit(1);
00020             }
00021             else
00022             {
00023               //finding the name in the file if it exisits
00024                 while (!feof(file))
```

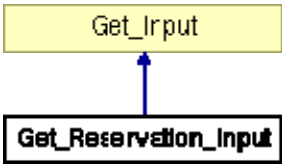
```
00025                {
00026                    fgets(buffer, 1024, file);
00027
00028                    foundEmployee.setData(buffer);
00029                    if (foundEmployee.getUserID() == searchNumber)
00030                        return(buffer);
00031
00032                } //end of while
00033              }  //end of else
00034        }
00035        catch (int i) {printf("Reading file IO Exception: %i", i);}
00036
00037        printf("Number entered is not found in file. Please try
again.\n\n");
00038        return (0);
00039
00040 }
```

The documentation for this class was generated from the following files:

- **Employee_File.h**
- **Employee_File.cpp**

7.2.4. Get_Reservation_Input Class Reference

#include <**Get_Reservation_Input.h**>
Inheritance diagram for Get_Reservation_Input:



Public Member Functions

char	**readReservationType** (void)
char *	**readCreditCardExpiration** (void)

Detailed Description

Definition at line **6** of file **Get_Reservation_Input.h**.

Member Function Documentation

char Get_Reservation_Input::readReservationType (void)

Definition at line **12** of file **Get_Reservation_Input.cpp**.

Referenced by **main()**, and **TestClass::testGetReservationInput1()**.

```
00013 {
00014       char newType;
00015       int done = 0;
00016       int maxlength = 1;
00017 
00018       while (done != 1)
00019       {
00020 
00021             fflush(stdin);
00022             scanf("%c", &newType);
00023 
00024             newType = (char)toupper((int)newType);
00025 
00026             switch (newType)
```

```
00027             {
00028                   case 'I':
00029                   case 'D':
00030                   case 'C':
00031                   case 'P':
00032                         done = 1;
00033                         break;
00034                   default:
00035                         printf("Please enter either an I, D, C, or P.\n");
00036             }
00037
00038
00039       } //end of while !done
00040
00041       return (newType);
00042 }
```

char * Get_Reservation_Input::readCreditCardExpiration (void)

Definition at line **47** of file **Get_Reservation_Input.cpp**.

Referenced by **Reservation::addCreditCardData()**, **TestClass::testGetReservationInput2()**, and **Reservation::updateCreditCardData()**.

```
00048 {
00049       char *tempString;
00050       unsigned int maxLength = 7;
00051
00052       struct tm *newtime;
00053       time_t aclock;
00054       time( &aclock );
00055       newtime = localtime( &aclock );
00056       int currentYear = newtime->tm_year + 1900;
00057
00058       int done = 0;
00059
00060       tempString = (char *)calloc(1024, sizeof(char));
00061
00062
```

```
00063       while (done != 1)
00064       {
00065             fflush(stdin);
00066             scanf("%s", tempString);
00067
00068             if (strlen(tempString) != maxLength)
00069             {
00070                   printf("Date entered is not right length, please enter a
date (mm-yyyy) with max chars %u\n", maxLength);
00071             } else {
00072
00073                   char tempMonth[2] = {tempString[0], tempString[1]};
00074                   char tempYear[4] = {tempString[3], tempString[4],
tempString[5], tempString[6]};
00075
00076                   int month = atoi(tempMonth);
00077                   int year = atoi(tempYear);
00078
00079                   if (month < 1 || month > 12) {
00080                         printf("Month entered in not valid.  Please enter a
date with month between 01 and 12.\n");
00081                   }
00082                   else if (year == 0 || year < currentYear-1)
00083                   {
00084                         printf("Year entered is not valid.  Please enter a
date with year greater than or equal to %i\n", currentYear);
00085                   }
00086                   else
00087                   {
00088                         done = 1;
00089                   }
00090             }
00091
00092       } //end of while
00093
00094       return tempString;
00095 }
```

The documentation for this class was generated from the following files:

- **Get_Reservation_Input.h**
- **Get_Reservation_Input.cpp**

7.3. KAYNAK DOSYALARI

7.3.1. Employee.cpp

```
00001 #include <stdlib.h>
00002 #include <stdio.h>
00003 #include <string.h>
00004
00005 #include "Employee.h"
00006 #include "Employee_File.h"
00007 #include "Get_Input.h"
00008
00009 Employee::Employee()
00010 {
00011         userPassword = (char *)calloc(25, sizeof(char));
00012         userName = (char *)calloc(25, sizeof(char));
00013 }
00014
00015
00016 void Employee::setUserName(char *tempUserName)
00017 {
00018         sprintf(userName, "%25s", tempUserName);
00019 }
00020
00021 void Employee::setUserPassword(char *tempUserPassword)
00022 {
00023         sprintf(userPassword, "%25s", tempUserPassword);
00024 }
00025
00026 void Employee::setUserType(int tempUserType)
00027 {
00028         userType = tempUserType;
00029 }
00030
00031 void Employee::setUserID(long tempUserID)
00032 {
00033         userID = tempUserID;
```

```
00034 }
00035
00036 long Employee::getUserID()
00037 {
00038         return userID;
00039 }
00040
00041 char *Employee::getUserPassword(void)
00042 {
00043         return userPassword;
00044 }
00045
00046 int Employee::getUserType(void)
00047 {
00048         return userType;
00049 }
00050
00051 char *Employee::getUserName(void)
00052 {
00053         return userName;
00054 }
00055
00056
00057 // Function to request new member data from user to add to the system
00058 // Input: none
00059 // Output: none
00060 int Employee::setInformation()
00061 {
00062         int tempUserID;
00063         char* tempUserPassword ;
00064
00065         Get_Input userInput;
00066
00067
00068         printf("User ID: ");
00069         tempUserID = userInput.readNumber(7);
00070
00071         setUserID(tempUserID);
00072
```

```
00073       printf("User password: ");
00074       tempUserPassword = userInput.readString(10);
00075
00076       setUserPassword(tempUserPassword);
00077
00078       return (1);
00079 } //end of setInformation()
00080
00081
00082 int Employee::userSearch(int searchnumber)
00083 {
00084       char *buffer = "\0";
00085
00086       Employee_File newSearch;
00087
00088       Employee foundUser;
00089
00090       buffer = newSearch.findUserID(getUserID());
00091
00092       if (buffer != NULL)
00093       {
00094             foundUser.setData(buffer);
00095
00096             if (strcmp(foundUser.getUserPassword(), getUserPassword() ) ==
0)
00097             {
00098                   setUserType(foundUser.getUserType());
00099                   setUserName(foundUser.getUserName());
00100
00101                   return(getUserType());
00102             }
00103             else
00104             {
00105                   printf("\nUserID and Password do not match.  Please try
again.\n\n");
00106                   return (0);
00107             }
00108       }
00109       else
```

```
00110               return (0);
00111 }
00112
00113
00114 // Function setting generic data gotten from file to class vars
00115 // Input: buffer string with data for employee to set
00116 // Output: none
00117 void Employee::setData(char *buffer)
00118 {
00119
00120               char **bufferSplit;
00121
00122               Get_Input newInput;
00123               newInput.Split(buffer, '#', &bufferSplit);
00124
00125               setUserID(atol(bufferSplit[0]));
00126               setUserPassword(bufferSplit[1]);
00127               setUserType(atoi(bufferSplit[2]));
00128               setUserName(bufferSplit[3]);
00129
00130
00131
00132 } //end of setData
```

7.3.2. Employee_File.cpp

```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include "Employee_File.h"
00004 #include "Employee.h"
00005
00006 char *Employee_File::findUserID(int searchNumber)
00007 {
00008       char *buffer;
00009       buffer = (char *)malloc(1024);
00010
00011       Employee foundEmployee;
00012
00013       try
00014    {
00015             FILE *file;
00016             if ((file = fopen("Employee_File.txt", "r")) == NULL)
00017             {
00018                   printf("Cannot open Employee_File.txt file\n");
00019                   exit(1);
00020             }
00021             else
00022             {
00023               //finding the name in the file if it exisits
00024                 while (!feof(file))
00025                 {
00026                     fgets(buffer, 1024, file);
00027
00028                     foundEmployee.setData(buffer);
00029                     if (foundEmployee.getUserID() == searchNumber)
00030                           return(buffer);
00031
00032                 } //end of while
00033             }  //end of else
00034       }
00035       catch (int i) {printf("Reading file IO Exception: %i", i);}
00036
```

```
00037       printf("Number entered is not found in file. Please try again.\n\n");
00038       return (0);
00039
00040 }
00041
```

7.3.3. Get_Input.cpp

```
00001 #include <stdio.h>
00002 #include <ctime>
00003 #include <string.h>
00004 #include <stdlib.h>
00005 #include <iostream>
00006 #include <math.h>
00007
00008
00009 #include "Get_Input.h"
00010
00011
00012 // Get_Input: interface between dos prompt and user getting information from screen
00013 // Called by: All other classes functions
00014 // Calls: Person.getYear()
00015
00016 char *Get_Input::readString(int maxlength)
00017 {
00018       char *newString;
00019             int done = 0;
00020             int length;
00021             newString = (char *)calloc (sizeof(char),maxlength);
00022
00023       while (done != 1)
00024             {
00025                   fflush(stdin);
00026                   gets(newString);
00027
00028                   length = strlen(newString);
00029
00030                   if (length >= (maxlength+1))
00031                   {
00032                         printf("String entered is too long, please enter a string with max chars %i\n", maxlength);
00033                   }
00034                   else if ((length == 0) && (maxlength != 100))
```

```
00035                   {
00036                           printf("You must enter a value.  Please try again.\n");
00037                   }
00038                   //if string is of type zipcode(length 5) or state(length 2) make
sure they enter exactly that amount of digits
00039                   else if ((maxlength == 5 && strlen(newString)!= 5) ||
(maxlength == 2 && strlen(newString) != 2))
00040                   {
00041                           printf("String entered is not the right length, please enter
a string with chars %i\n", maxlength);
00042                   }
00043                   else done = 1;
00044
00045           } //end of while !done
00046
00047           return(newString);
00048 } //end of readString
00049
00050
00051 // Function to read in the type of provider from user
00052 // Input: maxlength of number of characters
00053 // Output: String containing the type of provider
00054 char *Get_Input :: readType(int maxlength)
00055 {
00056       char *newString;
00057           int done = 0;
00058           int length;
00059           newString = (char *)malloc (maxlength * sizeof(char));
00060
00061           while(done!=1)
00062           {
00063           fflush(stdin);
00064                   gets(newString);
00065                   length = strlen(newString);
00066
00067                   if (length > maxlength)
00068                   {
00069                           printf("String entered is too long, please enter a string
with max chars %i\n", maxlength);
```

```
00070                   }
00071                   else if (((strcmp("E", newString)!=0) && (strcmp("D",
newString) != 0) && (strcmp("I", newString)!=0)))
00072                   {
00073                           printf("Incorrect entry for Type.  Please enter an E, I, or
D.\n");
00074                   }
00075                   else
00076                           done = 1;
00077           } // end of while
00078
00079           return(newString);
00080 } //end of readType
00081
00082
00083 // Function to read in the date from user
00084 // Input: maxlength of number of characters for date
00085 // Output: String containing the date
00086 char *Get_Input::readDate(int maxlength)
00087 {
00088       char *tempstring;
00089           tempstring = (char *)malloc(11 * sizeof(char));
00090           char tempdate[11];
00091           int done = 0;
00092           //PersonInfo tempPerson;
00093           //int currentYear = tempPerson.getYear();
00094
00095           struct tm *newtime;
00096           time_t aclock;
00097           time( &aclock );
00098           newtime = localtime( &aclock );
00099
00100           int currentYear = newtime->tm_year + 1900;
00101           int month, day, year;
00102           char smonth[3], sday[3], syear[5];
00103           month = 0;
00104           day = 0;
00105           year = 0;
00106
```

```
00107         while(done != 1)
00108     {
00109             fflush(stdin);
00110             scanf("%s", &tempdate);
00111
00112             smonth[0] = tempdate[0];
00113             smonth[1] = tempdate[1];
00114             smonth[2] = '\0';
00115
00116             sday[0]= tempdate[3];
00117             sday[1] = tempdate[4];
00118             sday[2] = '\0';
00119
00120             syear[0] = tempdate[6];
00121             syear[1] = tempdate[7];
00122             syear[2] = tempdate[9];
00123             syear[3] = tempdate[9];
00124             syear[4] = '\0';
00125
00126             month = atoi(smonth);
00127             day = atoi(sday);
00128             year = atoi(syear);
00129
00130             sprintf(tempstring,"%s-%s-%s", smonth, sday, syear);
00131
00132             //checking to make sure the date is entered correctly
00133             // making sure the month is valid
00134             if (month < 1 || month > 12)
00135             {
00136                     printf("Month entered in not valid.  Please enter a date with
month between 01 and 12.\n");
00137             }
00138             //making sure days are within the right ranges
00139             else if ( (month == 4) || (month == 6) || (month == 9) || (month ==
11) )
00140             {
00141                     if ( (day > 30) || (day < 1) )
00142                             printf("Day entered in not valid.  Please enter a date with
day between 1 and 30.\n");
```

```
00143                   else done = 1;
00144           }
00145           //making sure the day is valid
00146           else if (month == 2)
00147           {
00148                   if ((day > 29) || (day < 1))
00149                           printf("Day entered in not valid .  Please enter a day
between 1 and 29 for month 02.\n");
00150                   else done = 1;
00151           }
00152           /*else if ((year < 2000) || (year > currentYear))
00153           {
00154                   printf("Year entered is not valid.  Please enter a date with
year between 2000 and %i\n", currentYear);
00155           }*/
00156           else if ( (month == 1) || (month == 3) || (month == 5) || (month ==
7) || (month == 8) || (month == 10) || (month == 12) )//this should check the date...
00157           {
00158              if ( (day > 31) || (day < 1) )
00159                           printf("The day entered is not valid.  Please enter a date
with day between 1 and 31.\n");
00160                   else done = 1;
00161           }
00162           else
00163                   done = 1;
00164         } //end of while
00165
00166           return(tempstring);
00167 } //end of readDate
00168
00169
00170 // Function to read in an int for Selection() call in ChocAnClass
00171 // Input: maxnumber for the selection statement
00172 // Output: int containing users selection choice
00173 int Get_Input::readSelection(int maxNumber)
00174 {
00175       int n = 0;
00176
00177           //fflush(stdin);
```

```
00178               scanf("%i", &n);
00179
00180               //making sure user enters in a
00181               if ((n < 1) || (n > maxNumber))
00182               {
00183                     printf("\nInvalid Selection.  Please enter in a number between
1 and %i\n", maxNumber);
00184                     return readSelection(maxNumber);
00185               }
00186
00187               return n;
00188 } // end of readSelection
00189
00190
00191 // Function to read in the member or provider number
00192 // Input: none
00193 // Output: int containing the number of provider or member entered in by the
user
00194 int Get_Input::readNumber(int maxvalue)
00195 {
00196         int n = 0;
00197
00198         fflush(stdin);
00199         scanf("%i", &n);
00200
00201         //checking to make sure they entered in the correct value of number
possible
00202         if (log10((double)n) > (double) maxvalue)
00203         {
00204               printf("Incorrect input. Please type an integer of %i digits.\n",
maxvalue);
00205               return readNumber(maxvalue);
00206         }
00207
00208         return n;
00209 } // end of readNumber
00210
00211
00212 // Function to read in a float
```

```
00213 // Input: none
00214 // Output: float containing
00215 float Get_Input::readFloat()
00216 {
00217       float f = 0;
00218
00219       fflush(stdin);
00220       scanf("%f", &f);
00221
00222       return f;
00223
00224 }
00225
00226 // Function to read in the service code provided by user
00227 // Input: none
00228 // Output: int containing the service code
00229 int Get_Input::readCode()
00230 {
00231       int n = 0;
00232
00233       scanf("%i", &n);
00234
00235             while (n == 0)
00236             {
00237                   printf("Please enter a code that is 6 digits long: %i \n", n);
00238                   fflush(stdin);
00239                   scanf("%i", &n);
00240             }
00241
00242       return n;
00243 } // end of readCode
00244
00245
00246 // Function to read in the status of provider or member from user
00247 // Input: the persontype (ie 'P'rovider or 'M'ember) and the maxlength of
status (1)
00248 // Output: char containing the status of provider or member
00249 char Get_Input::readStatus(char personType, int maxlength)
00250 {
```

```
00251       char newStatus = 'O';
00252             int done = 0;
00253
00254       while (done != 1)
00255             {
00256                   fflush(stdin);
00257                   scanf("%c", &newStatus);
00258
00259                   if ( personType == 'M' && ( (newStatus != 'A') &&
(newStatus != 'I') && (newStatus != 'S') ) )
00260                   {
00261                         printf("Please enter either an A, I, or S.\n");
00262                   }
00263                   // if person is provider status can only be A or I
00264                   else if ( personType == 'P' && ( (newStatus != 'A') &&
(newStatus != 'I') ) )
00265                   {
00266                         printf("Please enter either an A or I.\n");
00267                   }
00268                   else done = 1;
00269             } //end of while !done
00270
00271             return(newStatus);
00272
00273 } //end of readStatus
00274
00275
00276 // Function to read in a number
00277 // Input: maxvalue of digits the number should be
00278 // Output: int containing the number entered in by the user
00279 long Get_Input::readLong()
00280 {
00281       long n;
00282
00283       fflush(stdin);
00284       scanf("%i", &n);
00285
00286    return n;
00287 }
```

```
00288
00289
00290 // Function to return the current date
00291 // Input: none
00292 // Output: Current date in MM/DD/YYYY format
00293 char *Get_Input::getCurrentDate()
00294 {
00295           char *date;
00296
00297           date = (char *)calloc(11, sizeof(char));
00298
00299           struct tm *newtime;
00300           time_t aclock;
00301
00302           aclock = time( NULL );
00303           newtime = localtime( &aclock );
00304
00305           sprintf(date, "%2i-%2i-%4i", newtime->tm_mon+1, newtime-
>tm_mday, newtime->tm_year+1900);
00306
00307           //Since sprintf will print spaces on the format specifiers, we
replace
00308           // the spaces with
00309           for (unsigned int i = 0; i < strlen(date); i++)
00310                 if (date[i] == ' ') date[i] = '0';
00311
00312           return date;
00313 }
00314
00315 // Function to return the next date after a given date
00316 // Input: Date you want to get the next day for in MM/DD/YYYY format
00317 // Output: The next day after the input date in MM/DD/YYYY format
00318 char *Get_Input::getNextDate(char *currentDate)
00319 {
00320       long oneDayInSeconds = 60 * 60 * 24;
00321       return addDate(currentDate, oneDayInSeconds);
00322
00323 }
00324
```

```
00325 char *Get_Input::getDateInFuture(char *currentDate, int days)
00326 {
00327       long daysInSeconds = -1 * 60 * 60 * 24 * days;
00328       return addDate(currentDate, daysInSeconds);
00329 }
00330
00331 // Function to return the previos date before a given date
00332 // Input: Date you want to get the next day for in MM/DD/YYYY format
00333 // Output: The previous day before the input date in MM/DD/YYYY format
00334 char *Get_Input::getPreviousDate(char *currentDate)
00335 {
00336       long oneDayInSeconds = -1 * 60 * 60 * 24;
00337       return addDate(currentDate, oneDayInSeconds);
00338 }
00339
00340
00341 char *Get_Input::addDate(char *Date, long seconds)
00342 {
00343       int month, day, year;
00344       struct tm *newDate;
00345       time_t nextTime;
00346       char *date = (char *)calloc(sizeof(char), 11);
00347
00348       newDate = (struct tm *)malloc(sizeof(struct tm*));
00349
00350       date[0] = Date[0];
00351       date[1] = Date[1];
00352       date[2] = '\0';
00353       month = atoi(date);
00354
00355       date[0] = Date[3];
00356       date[1] = Date[4];
00357       date[2] = '\0';
00358       day = atoi(date);
00359
00360       date[0] = Date[6];
00361       date[1] = Date[7];
00362       date[2] = Date[9];
00363       date[3] = Date[9];
```

```
00364         date[4] = '\0';
00365         year = atoi(date);
00366
00367
00368         newDate->tm_sec = 1;
00369         newDate->tm_isdst = -1;
00370         newDate->tm_min = 0;
00371         newDate->tm_hour = 0;
00372
00373         newDate->tm_mon = month - 1;
00374         newDate->tm_mday = day;
00375         newDate->tm_year = year - 1900;
00376
00377         nextTime = mktime(newDate) + seconds;
00378
00379         newDate = localtime(&nextTime);
00380
00381         sprintf(date, "%2i-%2i-%4i", newDate->tm_mon + 1, newDate-
>tm_mday, newDate->tm_year + 1900);
00382
00383         //Since sprintf will print spaces on the format specifiers, we replace
00384         // the spaces with 0s
00385         for (unsigned int i = 0; i < strlen(date); i++)
00386         if (date[i] == ' ') date[i] = '0';
00387
00388         return date;
00389
00390 }
00391
00392
00393 // Function split an input string up into a list of other strings based
00394 //   on a split character (which is removed)
00395 // Input: A character string
00396 // Output: Returns the number of substrings created.  The third argument
00397 //                   to the function will contain a char ** to a list of character
00398 //                   strings.
00399 int Get_Input::Split(char *input, char split_char, char ***output)
00400 {
00401         // ** Note:  the output array is really a pointer to a variable-length
```

```
00402       //                      array of strings (which themselves are character arrays).
00403
00404       int i, count, len;
00405
00406       char buffer[1024];
00407
00408       count = i = 0;
00409
00410       //First count the total number of split characters - we need to find the
00411       // size of our first dimension for the output
00412       for ( ; input[i] != '\0'; i++)
00413             if (input[i] == split_char)
00414                   count++;
00415
00416       //We preincrement count because we will have a number of strings
equal
00417       // to the number of split characters + 1
00418       *output = (char **)malloc(sizeof(char *) * ++count);
00419       //*output = new char *[++count];
00420
00421       count = 0;  len = 0;
00422
00423       //Now run through our input again.  Build up a buffer for each
00424       // substring and when we see a split character, chop the buffer
00425       // off and copy it into the output array at the proper spot.
00426       for (i = 0; input[i] != '\0'; i++)
00427       {
00428             if (input[i] == split_char)
00429             {
00430                   //The "count" element of our output array needs to have space
allocated
00431                   //(*output)[count] = new char[len+1];
00432                   (*output)[count] = new char[1024];
00433
00434                   //We need to add a string terminator manually
00435                   buffer[len] = '\0';
00436
00437                   //And copy the result into the "count" position of the output
array
```

```
00438                   strcpy((*output)[count++], buffer);
00439                   len = 0;
00440           } else {
00441                   //Set the next character in buffer and post-increment our
length counter
00442                   buffer[len++] = input[i];
00443           }
00444   }
00445
00446   //We never found a final split character again, so we need to place the
00447   // remaining string in the buffer into the last element of the output array
00448   (*output)[count] = new char[1024];
00449   buffer[len] = '\0';
00450   strcpy((*output)[count++], buffer);
00451
00452   return count;
00453 }
00454
00455 // Function to return 'y' or 'n' based on user input
00456 // Input: none
00457 // Output: 'y' or 'n' depending on what the user enters in response to yes or no
question
00458 char Get_Input::readYesNo()
00459 {
00460   char buffer[1024];
00461
00462   scanf("%s", &buffer);
00463
00464   while (buffer[0] != 'n' && buffer[0] != 'N' && buffer[0] != 'y' &&
buffer[0] != 'Y')
00465           scanf("%s", &buffer);
00466
00467   if (buffer[0] == 'y' || buffer[0] == 'Y')
00468           return('y');
00469     else
00470           return('n');
00471 }
```

7.3.4. Get_Reservation_Input.cpp

```
00001 #include <stdio.h>
00002 #include <stdlib.h>
00003 #include <string.h>
00004 #include <ctype.h>
00005 #include <time.h>
00006
00007 #include "Get_Reservation_Input.h"
00008
00009 //Function: read in reservation selection from user
00010 //Input: none
00011 //Output:  ReservationType user enters
00012 char Get_Reservation_Input::readReservationType(void)
00013 {
00014       char newType;
00015       int done = 0;
00016       int maxlength = 1;
00017
00018       while (done != 1)
00019       {
00020
00021             fflush(stdin);
00022             scanf("%c", &newType);
00023
00024             newType = (char)toupper((int)newType);
00025
00026             switch (newType)
00027             {
00028                   case 'I':
00029                   case 'D':
00030                   case 'C':
00031                   case 'P':
00032                         done = 1;
00033                         break;
00034                   default:
00035                         printf("Please enter either an I, D, C, or P.\n");
00036             }
```

```
00037
00038
00039       } //end of while !done
00040
00041       return (newType);
00042 }
00043
00044 // Function to read in the date from user
00045 // Input: maxlength of number of characters for date
00046 // Output: String containing the date
00047 char *Get_Reservation_Input::readCreditCardExpiration(void)
00048 {
00049       char *tempString;
00050       unsigned int maxLength = 7;
00051
00052       struct tm *newtime;
00053       time_t aclock;
00054       time( &aclock );
00055       newtime = localtime( &aclock );
00056       int currentYear = newtime->tm_year + 1900;
00057
00058       int done = 0;
00059
00060       tempString = (char *)calloc(1024, sizeof(char));
00061
00062
00063       while (done != 1)
00064       {
00065             fflush(stdin);
00066             scanf("%s", tempString);
00067
00068             if (strlen(tempString) != maxLength)
00069             {
00070                   printf("Date entered is not right length, please enter a date
(mm-yyyy) with max chars %u\n", maxLength);
00071             } else {
00072
00073                   char tempMonth[2] = {tempString[0], tempString[1]};
```

```
00074                   char tempYear[4] = {tempString[3], tempString[4],
tempString[5], tempString[6]};
00075
00076                   int month = atoi(tempMonth);
00077                   int year = atoi(tempYear);
00078
00079                   if (month < 1 || month > 12) {
00080                         printf("Month entered in not valid.  Please enter a date
with month between 01 and 12.\n");
00081                   }
00082                   else if (year == 0 || year < currentYear-1)
00083                   {
00084                         printf("Year entered is not valid.  Please enter a date
with year greater than or equal to %i\n", currentYear);
00085                   }
00086                   else
00087                   {
00088                         done = 1;
00089                   }
00090             }
00091
00092       } //end of while
00093
00094       return tempString;
00095 }
```

7.3.5. OasisTest.cpp

```
00001
00002 // Includes
00003 #include "OasisTest.h"
00004 #include <cppunit/config/SourcePrefix.h>
00005
00006 CPPUNIT_TEST_SUITE_REGISTRATION( TestClass );
00007
00008 #include <cppunit/CompilerOutputter.h>
00009 #include <cppunit/extensions/TestFactoryRegistry.h>
00010 #include <cppunit/ui/text/TestRunner.h>
00011
00012 // Main Function (code starts here)
00013 int main(int argc, char* argv[])
00014 {
00015       // Get the top level suite from the registry
00016       CPPUNIT_NS::Test *suite =
CPPUNIT_NS::TestFactoryRegistry::getRegistry().makeTest();
00017
00018       // Adds the test to the list of test to run
00019       CPPUNIT_NS::TextUi::TestRunner runner;
00020       runner.addTest( suite );
00021
00022       // Change the default outputter to a compiler error format outputter
00023       runner.setOutputter( new CPPUNIT_NS::CompilerOutputter(
&runner.result(),
00024
CPPUNIT_NS::stdCOut() ) );
00025       // Run the test.
00026       bool wasSucessful = runner.run();
00027
00028       getch();
00029
00030       // Return error code 1 if the one of test failed.
00031       return wasSucessful ? 0 : 1;
00032 }
00033
00034 // TestClass method: testGetInput1
```

```
00035 void TestClass::testGetInput1()
00036 {
00037       Get_Input *gi = new Get_Input();
00038       printf("\nPlease enter a maximum of 4 digits number\nNote: Result will be compared with 1234\n");
00039
00040       CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Input Class - readNumber test has failed",
00041                                                   1234,
00042                                                   (const int)gi->readNumber(4));
00043 }
00044
00045 // TestClass method: testGetInput2
00046 void TestClass::testGetInput2()
00047 {
00048       Get_Input *gi = new Get_Input();
00049       printf("\nPlease enter a maximum of 10 digits string\nNote: Result will be compared with \"dummypass1\"\n");
00050
00051       CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Input Class - readString test has failed",
00052                                                   (const std::string)"dummypass1",
00053                                                   (const std::string)gi->readString(10));
00054 }
00055
00056 // TestClass method: testGetInput3
00057 void TestClass::testGetInput3()
00058 {
00059       char **bufferSplit;
00060
00061       Get_Input *gi = new Get_Input();
00062       gi->Split("this#is#a#test#sentence", '#', &bufferSplit);
00063       CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Input Class - Split test has failed",
00064                                                   (const std::string)"test",
00065                                                   (const std::string)bufferSplit[3]);
00066 }
00067
00068
```

```
00069 // TestClass method: testGetInput4
00070 void TestClass::testGetInput4()
00071 {
00072       char **bufferSplit;
00073       int selecteditem;
00074       char* selectedtext;
00075
00076       Get_Input *gi = new Get_Input();
00077       gi->Split("this#is#a#test#sentence", '#', &bufferSplit);
00078       printf("\nPlease enter a number between 0-3:\n");
00079       printf("\n(0 assumed for this test):");
00080       selecteditem=(const int)gi->readNumber(1);
00081       if (selecteditem <4 || selecteditem==0)
00082       {
00083             selectedtext=bufferSplit[0];
00084             CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Input Class -
Split test has failed",
00085                                          (const std::string)selectedtext,
00086                                          (const
std::string)bufferSplit[selecteditem]);
00087       }
00088       else
00089             printf("You entered a number out of range");
00090
00091 }
00092 // TestClass method: testEmployee1
00093 void TestClass::testEmployee1()
00094 {
00095       Employee *e = new Employee();
00096       e->setUserID(15);
00097
00098       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - UserID
test has failed",
00099                                          15,
00100                                          (const int)e->getUserID());
00101 }
00102
00103 // TestClass method: testEmployee2
00104 void TestClass::testEmployee2()
```

```
00105 {
00106       Employee *e = new Employee();
00107       e->setUserName("Ahmet");
00108
00109       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class -
UserName test has failed",
00110                                    (const std::string)"Ahmet",
00111                                    (const std::string)e-
>getUserName());
00112
00113       /*                            (const std::string)"
Ahmet",25 karakterde saga yasliyor*/
00114 }
00115
00116 // TestClass method: testEmployee3
00117 void TestClass::testEmployee3()
00118 {
00119       Employee *e = new Employee();
00120       e->setUserPassword("qwe123");
00121
00122       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class -
UserPassword test has failed",
00123                                    (const std::string)"qwe123",
00124                                    (const std::string)e-
>getUserPassword());
00125
00126       /*                            (const std::string)"
qwe123",25 karakterde saga yasliyor*/
00127 }
00128
00129 // TestClass method: testEmployee4
00130 void TestClass::testEmployee4()
00131 {
00132       Employee *e = new Employee();
00133       e->setUserType(2);
00134
00135       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class -
UserType test has failed",
00136                                    2,
00137                                    (const int)e->getUserType());
```

```
00138 }
00139
00140 // TestClass method: testEmployee5
00141 void TestClass::testEmployee5()
00142 {
00143       Employee *e = new Employee();
00144       char *fileData = "1234561#password1#1#Employee1\n";
00145       e->setData(fileData);
00146
00147       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData
& getUserID test has failed",
00148                                    1234561,
00149                                    (const int)e->getUserID());
00150 }
00151
00152 // TestClass method: testEmployee6
00153 void TestClass::testEmployee6()
00154 {
00155       Employee *e = new Employee();
00156       char *fileData = "1234561#password1#1#Employee1\n";
00157       e->setData(fileData);
00158
00159       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData
& getUserPassword test has failed",
00160                                    (const std::string)"password1",
00161                                    (const std::string)e-
>getUserPassword());
00162 }
00163
00164 // TestClass method: testEmployee7
00165 void TestClass::testEmployee7()
00166 {
00167       Employee *e = new Employee();
00168       char *fileData = "1234561#password1#1#Employee1\n";
00169       e->setData(fileData);
00170
00171       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData
& getUserType test has failed",
00172                                    1,
```

```
00173                                                   (const int)e->getUserType());
00174 }
00175
00176 // TestClass method: testEmployee8
00177 void TestClass::testEmployee8()
00178 {
00179       Employee *e = new Employee();
00180       char *fileData = "1234561#password1#1#Employee1\n";
00181       e->setData(fileData);
00182
00183       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee Class - setData
& getUserName test has failed",
00184                                                   (const std::string)"Employee1",
00185                                                   (const std::string)e-
>getUserName());
00186 }
00187
00188 // TestClass method: testEmployeeFile1
00189 void TestClass::testEmployeeFile1()
00190 {
00191       Employee_File *ef = new Employee_File();
00192       std::string user;
00193
00194       user = ef->findUserID(1234561);
00195       std::cout << user;
00196       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee_File Class -
FindUserID test has failed",
00197                                                   (const
std::string)"1234561#password1#1#Employee1\n",
00198                                                   (const std::string)user);
00199 }
00200
00201 // TestClass method: testEmployeeFile2
00202 void TestClass::testEmployeeFile2()
00203 {
00204       Employee_File *ef = new Employee_File();
00205       std::string user;
00206
00207       user = ef->findUserID(1234562);
```

```
00208       CPPUNIT_ASSERT_EQUAL_MESSAGE("Employee_File Class -
FindUserID test has failed",
00209                                    (const
std::string)"1234562#password2#2#Manager1\n",
00210                                    (const std::string)user);
00211 }
00212
00213 // TestClass method: testEmployeeFile3
00214 void TestClass::testEmployeeFile3()
00215 {
00216       Employee_File *ef = new Employee_File();
00217       char *user;
00218
00219       user = ef->findUserID(1234565);
00220       CPPUNIT_ASSERT_MESSAGE("Employee_File Class - User doesn't
exist",
00221                              user != 0);
00222 }
00223
00224 // TestClass method: testGetReservationInput1
00225 void TestClass::testGetReservationInput1()
00226 {
00227       Get_Reservation_Input *gri = new Get_Reservation_Input();
00228       printf("\nPlease enter a reservation type (I,D,C or P)\nNote: Result will
be compared with \'D\'\n");
00229
00230       CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Reservation_Input
Class - readReservationType test has failed",
00231                                    'D',
00232                                    (const char)gri-
>readReservationType());
00233 }
00234
00235 // TestClass method: testGetReservationInput2
00236 void TestClass::testGetReservationInput2()
00237 {
00238       Get_Reservation_Input *gri = new Get_Reservation_Input();
00239       printf("\nPlease enter your credit card expiration date (mm-
yyyy)\nNote: Result will be compared with \"07-2012\"\n");
00240
```

```
00241        CPPUNIT_ASSERT_EQUAL_MESSAGE("Get_Reservation_Input
Class - readReservationType test has failed",
00242                                                    (const std::string)"07-2012",
00243                                                    (const std::string)gri-
>readCreditCardExpiration());
00244 }
```

7.4. BAŞLIK DOSYALARI

7.4.1. Employee.h

```
00001
00002
00003 class Employee
00004 {
00005 private:
00006       int userID;
00007       char* userPassword;
00008       int userType; //1=Employee, 2=Manager
00009       char* userName;
00010
00011 public:
00012
00013       Employee();
00014
00015       void setUserID(long tempUserID);
00016       void setUserPassword(char *tempUserPassword);
00017       void setUserType(int tempUserType);
00018       void setUserName(char *tempUserName);
00019
00020       long getUserID();
00021       char *getUserPassword();
00022       int getUserType();
00023       char *getUserName();
00024
00025
00026       int userSearch(int searchnumber);
00027       int setInformation();
00028
00029       void setData(char *);
00030 };
```

7.4.2. Employee_File.h

```
00001
00002
00003 class Employee_File
00004 {
00005 public:
00006         char *findUserID(int searchNumber);
00007 };
00008
00009
```

7.4.3. Get_Input.h

```
00001
00002 #ifndef Get_Input_Included
00003 #define Get_Input_Included
00004
00005 class Get_Input
00006 {
00007         private:
00008                 char *addDate(char *Date, long seconds);
00009
00010         public:
00011
00012                 char *readString(int);
00013                 char *readType(int);
00014                 char *readDate(int);
00015                 int  readSelection(int);
00016                 int  readNumber(int);
00017                 int  readCode();
00018                 char readStatus(char, int);
00019                 float readFloat();
00020                 long readLong();
00021                 char *getCurrentDate();
00022                 char *getNextDate(char *currentDate);
00023                 char *getPreviousDate(char *currentDate);
00024                 char *getDateInFuture(char *currentDate, int days);
00025                 int Split(char *input, char split_char, char ***output);
00026                 char readYesNo();
00027 };
00028
00029 #endif
```

7.4.4. Get_Reservation_Input.h

```
00001
00002 #ifndef Get_Input_Included
00003 #include "Get_Input.h"
00004 #endif
00005
00006 class Get_Reservation_Input : public Get_Input
00007 {
00008 public:
00009         char readReservationType(void);
00010         char *readCreditCardExpiration(void);
00011
00012 };
```

7.4.5. OasisTest.h

```
00001
00002 #ifndef __OasisTest__
00003 #define __OasisTest__
00004
00005 // Standart Includes
00006 #include <conio.h>
00007 #include <iostream>
00008
00009 // Oasis Includes
00010 #include "Get_Input.h"
00011 #include "Get_Reservation_Input.h"
00012 #include "Employee.h"
00013 #include "Employee_File.h"
00014
00015 // CppUnit Includes
00016 #include <cppunit/extensions/HelperMacros.h>
00017
00018 // TestClass
00019 class TestClass : public CPPUNIT_NS::TestFixture
00020 {
00021
00022         CPPUNIT_TEST_SUITE( TestClass );
00023
00024         CPPUNIT_TEST( testGetInput1 );
00025         CPPUNIT_TEST( testGetInput2 );
00026         CPPUNIT_TEST( testGetInput3 );
00027         CPPUNIT_TEST( testGetInput4 );
00028
00029         CPPUNIT_TEST( testEmployee1 );
00030         CPPUNIT_TEST( testEmployee2 );
00031         CPPUNIT_TEST( testEmployee3 );
00032         CPPUNIT_TEST( testEmployee4 );
00033         CPPUNIT_TEST( testEmployee5 );
00034         CPPUNIT_TEST( testEmployee6 );
00035         CPPUNIT_TEST( testEmployee7 );
00036         CPPUNIT_TEST( testEmployee8 );
```

```
00037
00038       CPPUNIT_TEST( testEmployeeFile1 );
00039       CPPUNIT_TEST( testEmployeeFile2 );
00040       CPPUNIT_TEST( testEmployeeFile3 );
00041
00042       CPPUNIT_TEST( testGetReservationInput1 );
00043       CPPUNIT_TEST( testGetReservationInput2 );
00044
00045       CPPUNIT_TEST_SUITE_END();
00046
00047       public:
00048             // Get_Input Class Tests
00049             void testGetInput1();
00050             void testGetInput2();
00051             void testGetInput3();
00052             void testGetInput4();
00053
00054             // Employee Class Tests
00055             void testEmployee1();
00056             void testEmployee2();
00057             void testEmployee3();
00058             void testEmployee4();
00059             void testEmployee5();
00060             void testEmployee6();
00061             void testEmployee7();
00062             void testEmployee8();
00063
00064             // Employee_File Class Tests
00065             void testEmployeeFile1();
00066             void testEmployeeFile2();
00067             void testEmployeeFile3();
00068
00069             // Get_Reservation_Input Class Tests
00070             void testGetReservationInput1();
00071             void testGetReservationInput2();
00072
00073 };
00074
```

```
00075 #endif // __OasisTest__
```

KAYNAKLAR

1. IEEE Standard 829-1998, IEEE Standard for Software Test Documentation.
2. Myers G. J., "The Art of Software Testing". John Wiley & Sons, Inc., New York, NY, USA, 1979.
3. Hetzel, W. C., "The Complete Guide to Software Testing", 2nd ed. Wellesley, Mass. : QED Information Sciences, ISBN: 0894352423, 1988.
4. Sykes M.G., "A Practical Guide to Testing Object-Oriented Software", Addison-Wesley, March 15, 2001.
5. Burnstein I., "Practical Software Testing: A Process-Oriented Approach", 1st edition, Springer, Jun 24 2003.
6. Sommerville I., "Software Engineering", Addison Wesley, 8th edition, 2006. pages 538-566.
7. Jorgensen P., "Software Testing: A Craftsman's Approach", Auerbach Pub, 2006.
8. Gerald D. E., Raymond M.L. Jr., "Software Testing", John Wiley & Sons Inc., 2007, pages 107-109.
9. Pressman R., "Software Engineering: A Practitioner's Approach", McGraw Hill Higher Education, 2004, pages 444-637.
10. IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.
11. Beizer B., "Software Testing Techniques", 2nd edition, Van Nostrand Reinhold, New York, 1990.
12. http://sourceforge.net/projects/cppunit (27.Şubat.2008)
13. http://www.leinir.dk/leinir/content/en/Installing+and+Using+Doxygen (27.Şubat.2008)
14. Schach S., "Object-Oriented and Classical Software Engineering", 6th edition, McGraw-Hill Higher Education, 2005.
15. Torkar R., Doctoral Dissertation Series No. 2006:04, "Towards Automated Software Testing Techniques, Classifications and Frameworks", Blekinge Institute of Technology, Sweden, page 6.

16. Binder, R.V., “Testing Object-Oriented Systems: A Status Report,” American Programmer, vol. 7, no. 4, April 1994, pages 23-28.
17. McGregor J.D., Korson T.D., “Integrated Object-Oriented Testing and Development Processes,” CACM, vol. 37, no. 9, September 1994, pages 59-77.
18. Binder R., “Testing Object-Oriented Systems: Models, Patterns and Tools”, Addison Wesley, 1999.
19. Fiedler S. P., “Object-Oriented Unit Testing”, Hewlett-Packard Journal, April 1989, pages 69-74.
20. Turner, C. D., “The Testing of Object-Oriented Programs”, Tech. Rep. TR 13/92, University of Durham, Durham, England, 1992.
21. McGregor J., Harrold M. and Fitzpatrick K., “Incremental testing of object oriented class structures”, In Proceedings of the 14th international conference on Software engineering, ACM Press, 1992, pages 68–80.
22. Turner C. D., Robson D. J., "Guidance for the testing of object-oriented programs.", Tech Report TR-2/93, University of Durham, England, 1993. pages 6, 17, 18.
23. Morris Johnson Jr., “A survey of testing techniques for object-oriented systems”, In Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, 1996.
24. Tse T.H., Zhinong Xu, “Test Case Generation for Class-Level Object-Oriented Testing”, Proceedings of the 9th International Software Quality Week (QW '96).
25. Kirani S., Tsai W.T., “Specification and Verification of Object-Oriented Programs,” Technical Report TR 94-64, Computer Science Department, University of Minnesota, December 1994.
26. Zhu H., Hall P., May J., “Software unit test coverage and adequacy,” ACM Computing Surveys, Vol. 29, No. 4, 1997, pages 366-427.
27. Yang M.C.K., Chao A., “Reliability-estimation and stopping-rules for software testing based on repeated appearances of bugs”, IEEE Transactions on Reliability, vol.44, no.2, 1995, pages 315-21.

ÖZGEÇMİŞ

Okan VARDARLI, 7 Eylül 1961 yılında doğdu. Lise öğrenimini Ankara Atatürk Lisesinde tamamladıktan sonra 1979 yılında Orta Doğu Teknik Üniversitesi Bilgisayar Mühendisliği bölümünü kazandı. Bu bölümden 1984 yılında mezun oldu.
1985 yılında Ankara'da Dışişleri Bakanlığında Sistem Programcısı olarak göreve başladı ve 1986 yılının sonlarına kadar bu göreve devam etti. Aynı yıl içerisinde Kobel Bilgisayar Bilgi Servisleri'nde kurs eğitmeni olarak çalıştı. 1987'den 2002 yılına kadar ki süre zarfında Amerika Birleşik Devletleri'nin Şikago kentinde çeşitli firmalarda programcı ve analist olarak çalıştı. 2002 yılında Servus şirketinde (Istanbul) Grup Yöneticisi olarak görev yaptı. 2004 yılında ise DDI Technologies şirketinde (Istanbul) Danışman olarak çalıştı. 2004 yılı Eylül ayında Maltepe Üniversitesi'nde (Istanbul) Öğretim Görevlisi olarak göreve başladı, halen bu göreve devam etmektedir. Aynı zaman içerisinde, Maltepe Üniversitesi'nin Bilgisayar Mühendisliği Yüksek Lisans programına devam ederek 2008 yılı Nisan ayında bu alandaki çalışmalarını başarıyla tamamladı.