

HACETTEPE ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

JUNIT ÇATISI VE BİRİM TESTLERİ

PENBE MACİT
20022126
<penbe.macit@gmail.com>

İçindekiler

İçindekiler	1
JUnit çatısı ve Birim Testleri	2
1. Giriş	2
2. JUnit'in Kurulumu	2
3. JUnit'in Temelleri	5
4. Basit Bir Test	5
a. junit.framework paketi	7
b. TestCase Sınıfı	7
c. setUp() ve TearDown() Metodları	7
d. testBankahesabi(), testParaYatir() vs.	7
e. assertXXX() Türündeki Metodlar	7
5. Genel Çözümler	9
6. Referanslar	12
7. Ekler	13
a. BankaHesabi Sınıfı Kaynak Kodu	13
b. TestBankaHesabi Sınıfı Kaynak Kodu	14

JUNIT ÇATISI VE BİRİM TESTLERİ

1. Giriş

Bütün kodlar test edilir.

Geliştirme sırasında, kendi kabul edilebilir testlerimiz ile kodlarımızı deneriz. Önce kodlarımız, sonra derleriz ve en sonunda çalıştırırız. Çalıştırdığımız zaman ise test etmiş oluruz. Bir butona bastığımızda beklediğimiz menünün çıkması, hesaplama işlemlerinde doğru sonucun üretiliyor olması gibi sonuçlar programın testi geçtiğini gösterir. Fakat programa her ekleme yapıldığında derleme, çalıştırma ve test etme işlemlerinin hepsi tekrar tekrar yapılmak zorundadır. Programcıların çoğu yazdığı kodları kendi testleri ile denerler. Bunlar bir kayıt ekleme, silme, güncelleme veya listeleme gibi işlemlerdir. Bunların boyutunun fazla büyük olmadığı durumlarda, elle bu testleri yapmak zor olmayabilir ve defalarca yapılabilir. Bazı programcılar testlerini bu şekilde yapmayı severler. Çünkü bu onlar için kodlamaya, derin ve yorucu düşünmeye verilen bir aradır. Fakat diğer programcılar testlerini bu şekilde yapmazlar. Test verilerini elle girmek yerine küçük bir program geliştirip, programı otomatik olarak test ederler.

JUnit Java'da test işlemlerini gerçekleştirmek için kullanılan bir çatıdır(*framework*)^[1] ve Java programlarının özel alanlarının açıkça test edilmesi için kolay bir yol sunar. Genişleyebilir ve bir programın bir çok birimini test etmek için görevlendirilebileceği gibi tek bir biriminin testi için de görevlendirilebilir.

Bir test çatısı kullanmak kazançlı bir iştir, çünkü bu bizi beklenen sonuçları üreten program rutinlerini açıkça tanımlamaya zorlar. Programlarda hata ayıklaması(*debug*) yaparken, üretilmek istenen sonuçların açıklandığı bir test yazılabilir ve bu test olumlu sonuç verene kadar program kontrol edilebilir. Bir projenin en önemli kesimlerini oluşturan birimler için bir test setine sahip olmak, bu birimler üzerinde herhangi bir değişiklik yapıldığında bunların etkilerini test sonuçlarından hemen görmeyi sağlar ve yan etkilerinin hemen fark edilmesine yardımcı olur.

JUnit önce test etmeyi ve sonra kodlamayı destekler. Bu sayede programın belirli bir birimi için test verisi hazırlanabilir ve bu birim testi geçene kadar kodlama yapılabilir. Böyle hareket edilmesi programcının verimliliğini ve kodun kararlılığını artırır. Kodun kararsız olması programcının stresinin çoğalmasına ve kontrol için geçen sürenin artmasına neden olur.

Bu dokümanın kalan kesiminde, JUnit ile test geliştirmek için başlangıç düzeyinde bilgi verilecektir.

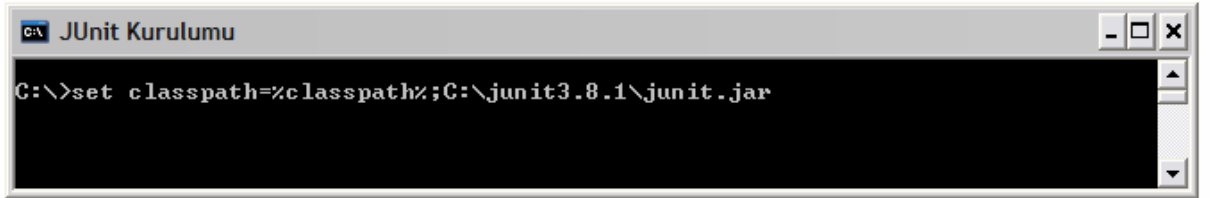
2. JUnit'in Kurulumu:

JUnit kurulum aşamaları aşağıdaki gibidir:

1. Programın son versiyonunu <http://download.sourceforge.net/junit/> adresinden temin edebilirsiniz.
2. junit.zip kütüğünün içeriğini bilgisayarınızda bir klasör altına çıkartın. (örneğin Windows için C:\ veya UNIX için /opt/)

^[1]Çatı (*framework*)—Yarı-tamamlanmış bir uygulamadır. Uygulamalar arasında paylaşılabilen yeniden kullanılabilir, ortak bir yapı sunar. Geliştiriciler kendi programları içine bir çatı yerleştirebilir ve bunu kendi ihtiyaçlarını karşılamak için genişletebilirler. Çatılar, basit ve yararlı sınıf setleri sunan araçların (*toolkit*) aksine daha yararlı olabilecek yapılar sunarlar.

3. **junit.jar** kütüğünün yolunu CLASSPATH değişkenine aktarın. Bu işlemi komut satırından aşağıdaki gibi yapabilirsiniz (Windows için):
- o set classpath=%classpath%;INSTALL_DIR\junit3\junit.jar



```
C:\>set classpath=%classpath%;C:\junit3.8.1\junit.jar
```

4. Yukarıdaki işlemleri yaptıktan sonra, kurulumunuzu JUnit sürümüyle birlikte gelen test kütükleri ile test edin. Tüm testlerin başarıyla sonuçlanması kurulumunuzu doğru yaptığınızı gösterir.

Not: Tüm testler junit.jar içinde mevcuttur. Bu yüzden CLASSPATH değişkenine doğru bir değer aktardığınızdan emin olunuz.

Testleri komut satırından aşağıdaki gibi çalıştırabilirsiniz:

- o Metin olarak TestRunner çalıştırmak için:

Windows:

Eğer CLASSPATH değişkenine aktarma yapamadıysanız, JUnit'in kurulu olduğu klasöre giderek aşağıdaki biçimde çalıştırın:

```
java -cp junit.jar;. junit.swingui.TestRunner junit.samples.AllTests
```

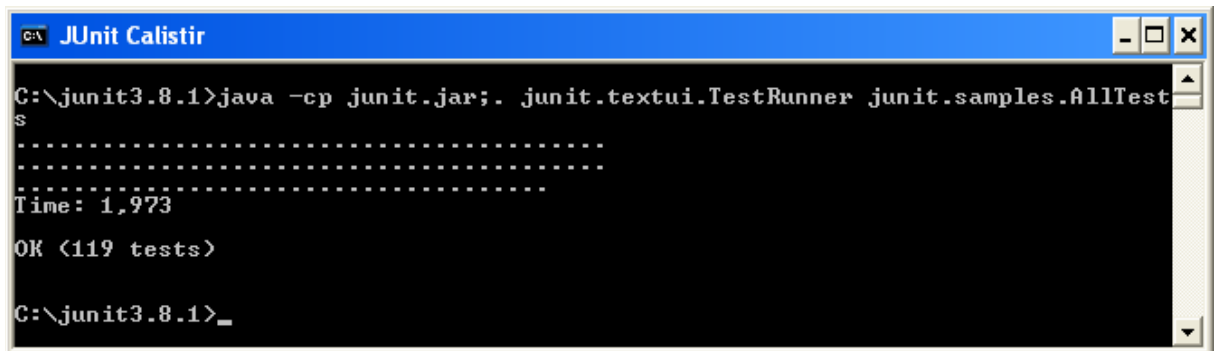
CLASSPATH değişkenine aktarma yapıldı ise :

```
java junit.textui.TestRunner junit.samples.AllTests
```

UNIX:

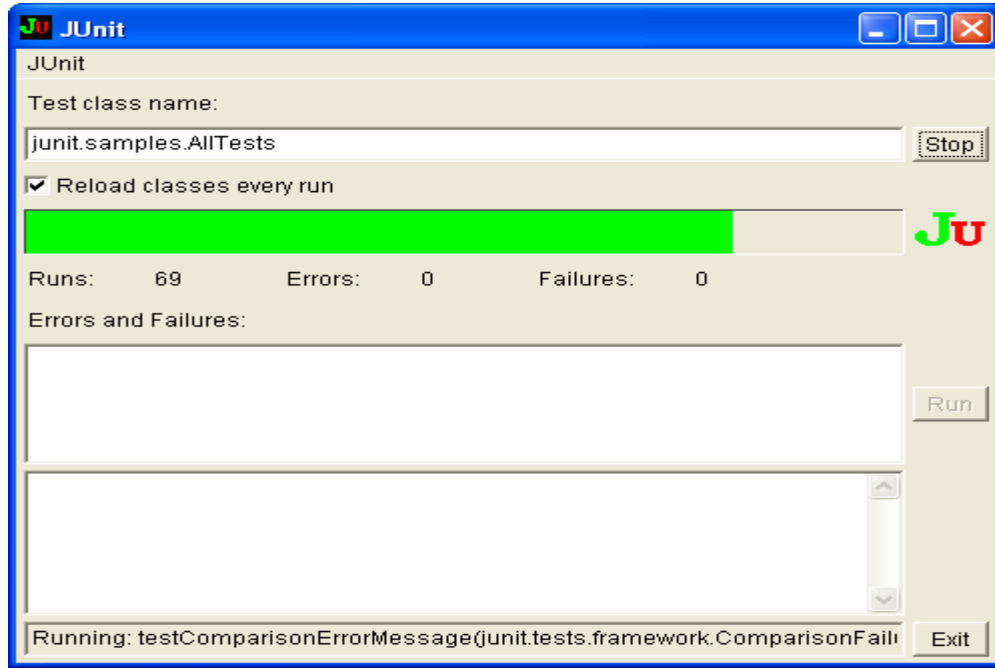
/opt/ altında aşağıdaki komutu çalıştırın:

```
java -cp junit.jar;. junit.swingui.TestRunner junit.samples.AllTests
```

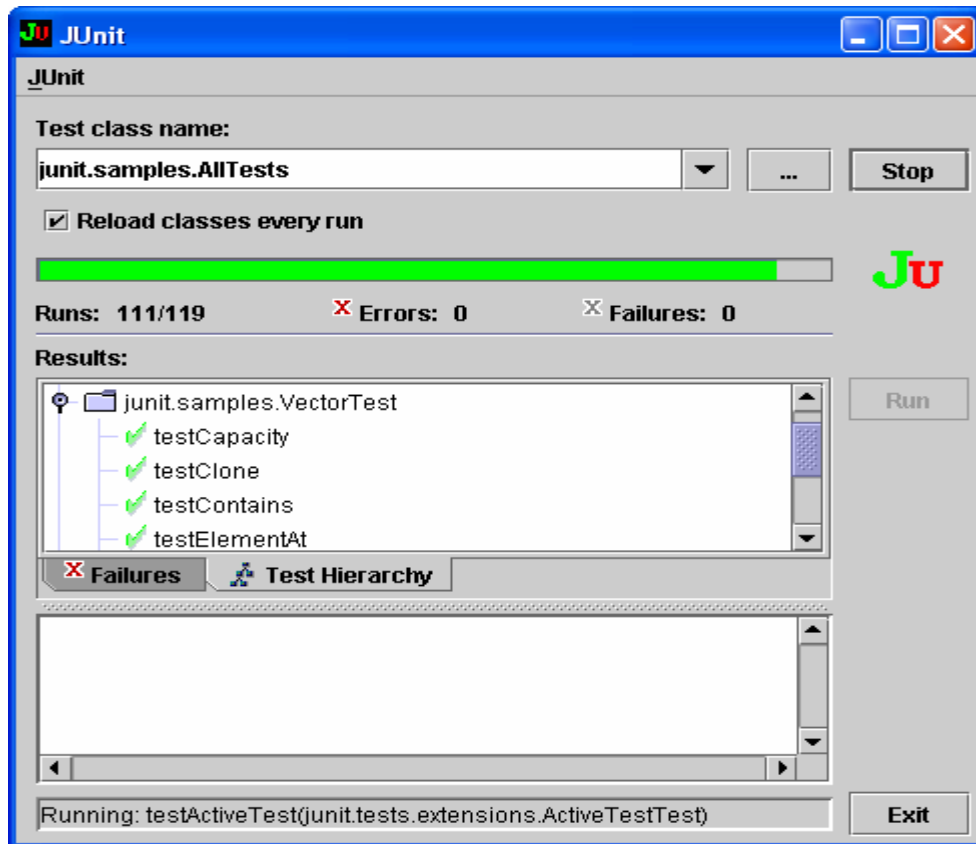


```
C:\junit3.8.1>java -cp junit.jar;. junit.textui.TestRunner junit.samples.AllTests
.....
.....
Time: 1.973
OK <119 tests>
C:\junit3.8.1>_
```

- Grafik olarak TestRunner çalıştırmak için:
java junit.awtui.TestRunner junit.samples.AllTests



- Swing tabanlı grafik olarak TestRunner çalıştırmak için:
java junit.swingui.TestRunner junit.samples.AllTests



3. JUnit'in Temelleri

JUnit test yazmayı ve çalıştırmayı kolaylaştıran çeşitli özelliklere sahiptir. Bunlar aşağıda özetlenmiştir:

- Test sonuçlarını görüntülemeyi sağlayan çeşitli arayüzlere sahiptir.
- Programın her biriminin(*unit*) ayrı test edilmesi için programı parçalara ayırır. Böylece bir birim üzerindeki etkiler diğer birimleri etkilemez. Ayrıca hataları bütün üzerinde aramaya gerek kalmadan küçük bir kesimde aranmasını sağlar. Böylece hatanın bulunması kolaylaşır.
- Standard yaratma ve temizleme metodları içerir(setUp ve tearDown).
- Sonuçları kontrol etmeyi kolaylaştıran metodlara sahiptir.
- Ant, Maven gibi araçlarla ve Eclipse, IntelliJ ve JBuilder gibi Bütünleşmiş Geliştirme Ortamları(Integrated Development Environment-IDE) ile bütünleştirilebilir.

Bu belgede, öncelikle program birimlerinin test edilmesi söz konusu edilecektir. Bir birim testi (*unit test case*), programınızın bir bölümünün davranışını doğrulamak için oluşturulmuş testleri içerir. Java'da, genellikle her program bölümü için bir sınıf oluşturulur. Yani bir birim testi, bir sınıfı test eder.

Testlerin genel bir yapısı vardır. Bir test yazmaya başladığında genellikle aşağıdaki sıra takip edilir:

1. Test edilecek sınıfların nesnelere yaratılır.
2. Bu nesnelere mesajlar gönderilir.
3. Alınan sonuçlar ile beklenen sonuçlar karşılaştırılır.

4. Basit Bir Test

JUnit'i daha açık olarak açıklamak için bu bölümde basit bir örnek yapılacaktır. Örnek olarak bir banka hesabı sınıfı yaratılacaktır. Bu banka hesabı müşterilerinin hesap numarasını, adını soyadını, bakiyesini ve hesabın o an aktif olup olmadığını tutacaktır.

```
class BankaHesabi {
    int bakiye=0;
    String hesapNo0null;
    String musterAdiSoyadi = null;
    boolean acik = false;
    public BankaHesabi(String hesapNo, String musterAdiSoyadi) {
        . . . . .
    }
    public String paraYatir(int miktar) {
        . . . . .
    }
    public String paraCek(int miktar) {
        . . . . .
    }
}
```

²Kaynak kodun tamamı 6. bölümünde "Ekler" başlığı altında incelenebilir. Kod burada, yalnızca genel bir fikir vermek amacıyla örneklendirilmiştir.

Banka hesabının çeşitli özellikleri³ sağlanması gerekmektedir. Bunlar:

- İlk yaratıldığında bakiyesi “0” olacaktır.
 - Para Yatırılincaya kadar aktif olmayacaktır.
 - Para çekilmek istendiğinde, çekilmek istenen miktar bakiyeden fazla olmamalıdır.
 - Bakiye sıfırlandığında hesap pasif duruma geçirilecektir.
 - Aktif olmayan hesaptan para çekilemez.
- gibi özelliklerdir.

Yukarıdaki sınıfı(BankaHesabi) yazan bir kişi ilk bakışta gerekli görülen özellikleri sağladığını düşünebilir fakat gözden kaçırdığı hatalar olabilir. JUnit’de bu hataları basit bir biçimde bulabilecek bir test sınıfı yazılabilir. Aşağıdaki örnek kod bu amaçla yazılmıştır:

```
import junit.framework.*;

public class TestBankaHesabi extends TestCase {
    private BankaHesabi bankaHesabi = null;

    protected void setUp() throws Exception {
        . . . . .
    }

    protected void tearDown() throws Exception {
        . . . . .
    }

    public void testParaYatir() {
        int miktar = 100;
        String beklenenSonuc = "Hesabiniza "+miktar+ " YTL yatırılmıştır";
        String gercekSonuc = bankaHesabi.paraYatir(miktar);
        assertEquals("Para yatırma", beklenenSonuc, gercekSonuc);
    }

    public void testBankaHesabi() {
        String hesapNo = "123456789";
        . . . . .
        assertEquals("Hesap Numarasi", hesapNo, bankaHesabi.hesapNo);
        . . . . .
        assertFalse("Acik mi", bankaHesabi.acik);
    }

    public void testParaCek() {
        int miktar = 10;
        String beklenenSonuc0 = "Hesabinizdan " +miktar+" YTL çekilmiştir";

        bankaHesabi.paraYatir(100);
        String gercekSonuc = bankaHesabi.paraCek(miktar);
        assertEquals("Para Çekme", beklenenSonuc0, gercekSonuc);
        . . . . .
    }
}
```

³Banka Hesaplarının gerçekte bu özelliklere sahip olması gerekmez. Bunlar, yalnızca bu örnek için tanımlanmış özelliklerdir

Yukarıdaki örnekte kullanılan çeşitli paketlerin ve metodların kullanım gerekçeleri aşağıdaki gibidir:

- a. **junit.framework paketi:** JUnit çatısının çeşitli sınıflarını ve metodların kullanabilmek için bu paketin kodumuzun içine dahil edilmesi gerekmektedir.
- b. **TestCase sınıfı:** Yazdığımız test sınıfının kullanılması için, kendi test metodlarını tanımlaması gerekmektedir. Bu nedenle, yazılacak test sınıfı, kendi test metodlarını tanımlama imkanı veren TestCase sınıfının bir alt sınıfı olmak durumundadır.
- c. **setUp() ve TearDown() metodları:** Her test metodu içinde test etmek için nesnelere yaratmamız gereklidir. Yaratılan bu nesnelere mesajlar yollar ve aldığımız yanıtlara göre yazdığımız kodun hatalı olup olmadığını test ederiz. Bu nesnelere yaratılması ve temizlenmesi işinin her test metodu içinde aynı olacağı açıktır. Bu işlerin yapıldığı kod kesimlerinin birden fazla yerde olmasını engellemek için setUp() ve tearDown() metodları kullanılabilir. setUp() metodu nesnelere yaratılmasında, tearDown() metodu da temizlenmesinde kullanılır. JUnit her test metodu için yeni bir test nesnesi yaratır. Yani üç test metodunuz varsa, JUnit test sınıfınızın üç tane olgusunu(nesne) yaratır. Bu olguların oluşturulmasından sonra aşağıdaki sıra takip edilir:

- setUp() metodu çağrılır.
- Test metodu çağrılır.
- tearDown metodu çağrılır.

Bu işlemler Test sınıfı içindeki her test metodu için tekrarlanır.

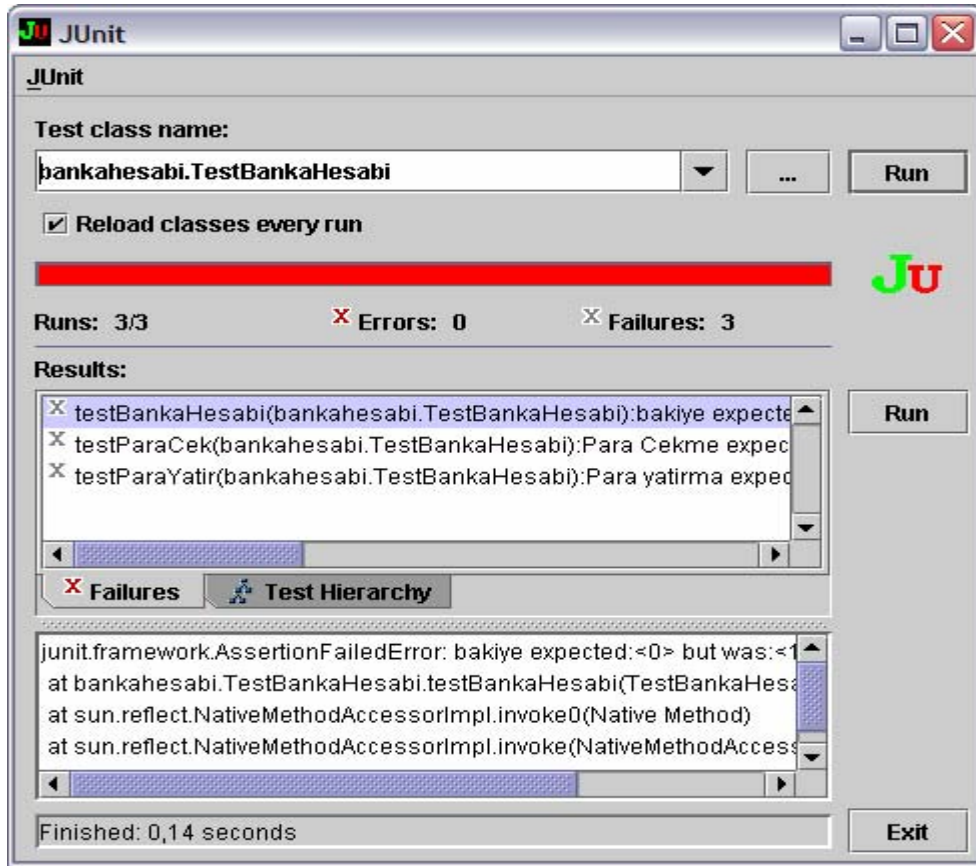
Çoğu zaman tearDown() metodu gözardı edilebilir çünkü kişisel birim testleri uzun süren işlemler içermezler ve nesnelere, Java Sanal makinasının(*Java Virtual Machine-JVM*) sonlandırılmasına yakın, çöp toplayıcısı(*garbage collector*) tarafından toplanırlar. Eğer test kodu veri tabanı bağlantıları açıyorsa, kullanıcı arayüzü sunuyorsa veya sistem kaynaklarının acilen serbest bırakılması isteniyorsa tearDown() metodu daha kullanışlı olabilir. Eğer geniş birim testleri yapıyorsanız tearDown() metodu içinde nesnelere “null” değerinin atanması çöp toplayıcısına, diğer testlere geçilmeden önce belleğin boşaltılmasında yardımcı olabilir. Yaratma işlemi, setUp() metodu kullanılmadan sınıfın yapıcı metodu(*constructor*) içinde de yapılabilir. Bu basit işlemlerde kabul edilebilir bir yoldur. Fakat sınıflarımız derin bir kalıtım yapısı içeriyorsa setUp() metodu kullanmak daha avantajlıdır. Bu sayede türetilen sınıfların tamamının yaratılması sonlanmadan ilk değer atamalarının yapılması önlenir.

- d. **testBankaHesabi(), testParaYatir() vs.:** Bu metodlar bizim sınıfımızı test etmek amacıyla yazacağımız asıl metodlardır. Bu metodlar JUnit tarafından otomatik olarak çağrılırlar. Her metod için ayrı bir test durumu (*test case*) yaratılır ve bu metodlar işletilir.
- e. **assertXXX() türündeki metodlar:** Bu metodlar JUnitin bize sunduğu ve değişik testleri yapmamızı sağlayan metodlardır. Örneğin banka hesabı kod örneğinde “testParaYatir” metodu içindeki assertEquals metodu, beklenen sonuç ile elde edilen gerçek sonucun aynı olup olmadığını test eder. Aynı şekilde “testBankaHesabi” metodu içindeki assertFalse metodu, hesabın ilk açıldığında pasif olup olmadığını kontrol eder. JUnit çatısı içinde bunlar gibi birçok assertXXX türünde metodlar vardır.^[4]

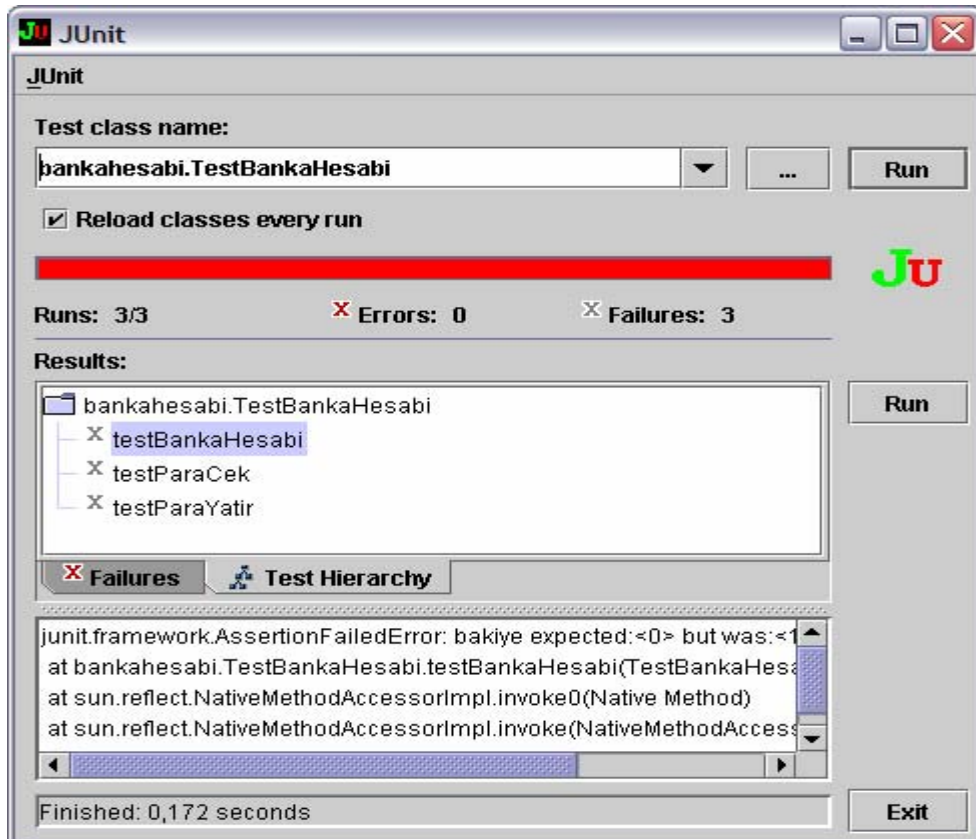
Yukarıdaki kod çalıştırıldığında^[5], eğer kodunuzda bir hata varsa aşağıdaki gibi bir ekran ile karşılaşırız:

^[4] Bu metodların ayrıntılı açıklamalarına JUnit ile birlikte gelen javadoc dökümanlarından ulaşabilirsiniz.

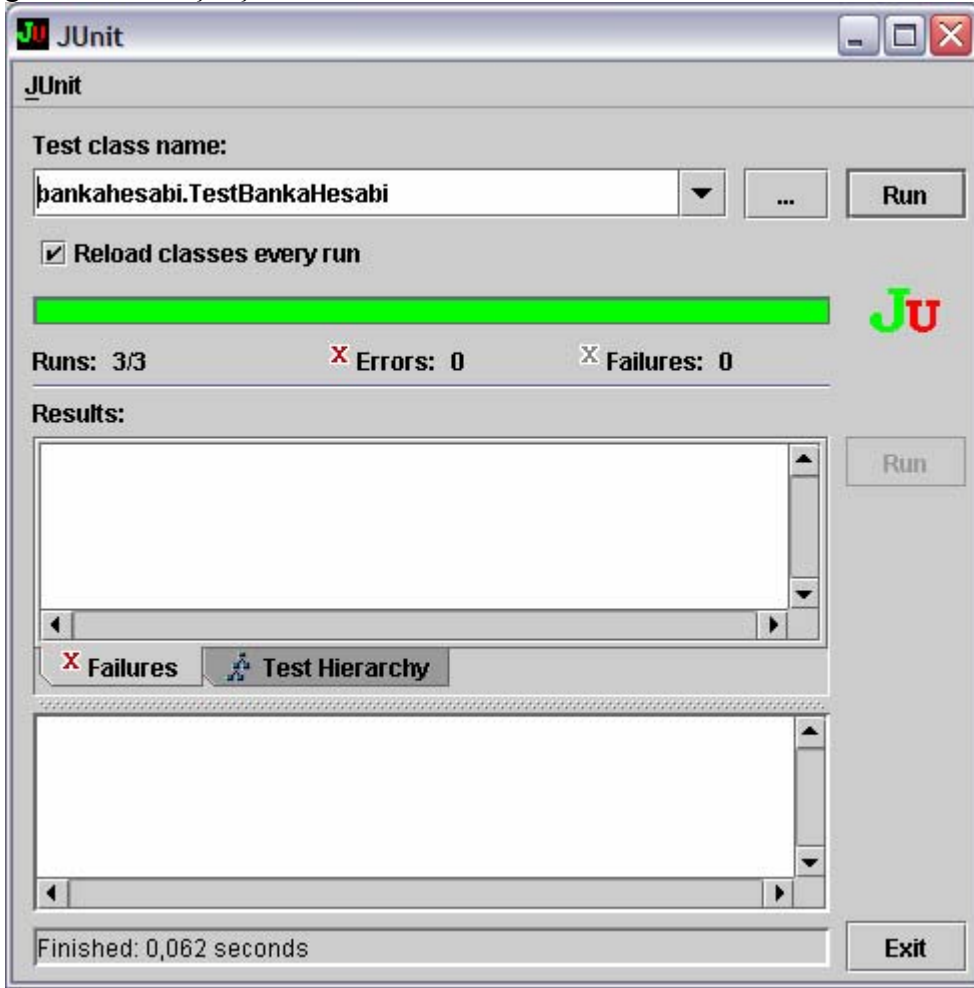
^[5] JUnit’in nasıl çalıştırılacağı, kurulum bölümünde, JUnit kurulumunu test etmek amacıyla açıklanmıştır.



Hangi test metodunun hataya sebep olduğunu “Test Hierarchy” görünümünde aşağıdaki gibi görebilirsiniz:



Burada özellikle hata yapılmıştır. Bu hataların içeriğini yukarıdaki gibi görebilir ve gerekli düzenlemeleri yapabilirsiniz. Eğer kodunuz testi başarıyla geçmiş ise aşağıdaki gibi bir görüntü ile karşılaşsınız:



5. Genel Çözümler

Bu bölümde JUnit ile yapılabilecek işlemlerin genel bir tanımı yapılacaktır.

- Tüm yaratma işlemini başta bir kere yapabilir ve bu işlem yapıldıktan sonra test kodlarını işletebilir ve sonunda bir kez temizleme kodunun çalıştırılmasını sağlayabilirsiniz. Bunu sağlamak için “junit.extensions.TestSetup” sınıfının kullanılması yeterlidir.

```
import junit.framework.*;
import junit.extensions.TestSetup;

public class TestBankaHesabi extends TestCase {
    . . . .
    public static Test suite() {
        TestSetup yarat = new TestSetup(new
            TestSuite(TestBankaHesabi.class)) {
            protected void setUp() throws Exception {....}
            protected void tearDown() throws Exception {....}
        };
        return yarat;
    }
}
```

- Bazı testlerin bir takım halinde işletilmesini sağlayabilirsiniz. Bunun için “junit.framework.TestSuite” sınıfını kullanabilirsiniz. JUnit, kodunuzun içinde “suite()” metodunu arar ve işletir.

```
import junit.framework.*;

public class TestBankaHesabi extends TestCase {
    . . . .
    public static Test suite() {
        new TestSuite(TestBankaHesabi.class);
    }
}
```

- Testlerinizi doğrudan çalıştırabilirsiniz. Bunu yapmak için test sınıfınıza bir “main” metodu ekleyebilirsiniz^[6].

```
import junit.framework.*;

public class TestBankaHesabi extends TestCase {
    . . . .
    public static void main(String args[]) {
        junit.textui.TestRunner.run(new TestSuite(TestBankaHesabi.class));
    }
}
```

- Testlerinizi istediğiniz miktarda çalıştırmak için “junit.extensions.RepeatedTest” sınıfını kullanabilirsiniz.

```
import junit.framework.*;
import junit.extensions.RepeatedTest;

public class TestBankaHesabi extends TestCase {
    . . . .
    public static Test suite() {
        return new RepeatedTest(new TestSuite(TestBankaHesabi.class), 100);
    }
}
```

- Kodunuzu test ederken doğru alanlarda doğru aykırı durumların fırlatıldığını da test edebilirsiniz.

```
public class TestBankaHesabi extends TestCase {
    . . . .
    public void testAykırıDurum() {
        try{
            . . . . //aykırı durum oluşmasını gerektiren işlemler yapılıyor

            fail("Aykırı durum oluşması bekleniyordu");
        }
        catch(BirAykırıDurum bak){
            //dogru aykırı durum fırlatıldığı için kod testi geçti
        }
    }
}
```

^[6]Testlerinizi Ant ile de çalıştırabilirsiniz. Ant ile ilgili bilgilere <http://ant.apache.org/manual/> adresinden ulaşabilirsiniz.

- İşletim dizileri(*threads*) kullanarak, birkaç testin eşanlı olarak çalıştırılmasını sağlayabilirsiniz. Bunu yapmak için “junit.extensions.ActiveTestSuite” sınıfını kullanabilirsiniz. Bu sınıf içine dahil edilmiş olan testlerin her birini ayrı bir işletim dizisi olarak işletir ve tüm diziler sonlanmadan suite metodu sonlandırılmaz.

```
import junit.framework.*;
import junit.extensions.RepeatedTest;

public class TestBankaHesabi extends TestCase {
    . . . .
    public static Test suite() {
        return new ActiveTestSuite(TestBankaHesabi.class);
    }
}
```

6. Referanslar

1. JUnit in Action, Vincent Massol with Ted Husted, ISBN 1-930110-99-5
2. JUnit Testing Utility Tutorial, Ashley J.S Mills, the University Of Birmingham
3. <http://www.teknoturk.org/docking/yazarlar/mehmet.kerem.kiziltunc.htm>
4. Java Extreme Programming Cookbook, O'Reilly, Eric M. Burke & Brian M. Coyner
5. JBuilderX Yardım Kütüphanesi.

7. Ekler

a. BankaHesabi Sınıfı Kaynak Kodu:

```
package bankahesabi;
public class BankaHesabi {
    /**
     * Banka Hesabında bulunan miktarı belirtir.
     * Bakiye 0 ise banka hesabi kapatılabilir.Aksi durumda hesap kapatılamaz.
     */
    int bakiye = 0;
    /**
     * Banka Hesabının biricik tanımlanmasını sağlayan degerdir.
     */
    String hesapNo = null;
    /**
     * Banka Hesabının ait olduğu müşterinin ad ve soyad bilgisini tutar.
     */
    String müşteriAdiSoyadi = null;
    /**
     * Banka Hesabının acik olup olmadığını belirtir.
     */
    boolean acik = false;
    /**
     * Yeni bir banka hesabi oluşturur.
     * @param hesapNo String Yaratılmak istenen banka hesabının biricik
     *   numarasını belirtir.
     * @param müşteriAdiSoyadi String banka hesabının ait olacağı müşterinin ad
     *   ve soyad bilgisini içerir.
     */
    public BankaHesabi(String hesapNo,String müşteriAdiSoyadi ) {
        this.hesapNo = new String (hesapNo);
        this.müşteriAdiSoyadi = new String (müşteriAdiSoyadi);
    }
    /**
     * Bakiyeye istenen miktarın eklenmesini sağlar
     * @param miktar int Banka Hesabına yatırılma istenen miktarı belirtir.
     * @return String Banka hesabına para yatırılıp yatırılmadığının
     *   açıklamasını döndürür.
     */
    public String paraYatir(int miktar){
        this.acik = true;
        this.bakiye += miktar;
        return "Hesabiniza "+miktar+ " YTL yatırılmıstır";
    }
    /**
     * Bakiyeden istenen miktarın düşülmesini sağlar
     * @param miktar int hesaptan çekilmek istenen miktarı belirtir.
     * @return String Hesaptan para çekilip çekilemedığının açıklamasını döndürür.
     */
    public String paraCek(int miktar){
        if(this.acik){
            if(this.bakiye >= miktar){
                this.bakiye -= miktar;
                if(this.bakiye==0)
                    this.acik = false;
                return "Hesabinizdan " +miktar+" YTL çekilmıstır";
            }
            else
                return "Hesabinızda yeterli miktar bulunmadığı için işleminiz yapılamamaktadır";
        }
        else{
            return "Acik olmayan bir hesaptan para çekemezsiniz";
        }
    }
}
```

b. TestBankaHesabi Sınıfı Kaynak Kodu:

```
package bankaHesabi;

import junit.framework.*;

public class TestBankaHesabi extends TestCase {
    private BankaHesabi bankaHesabi = null;

    protected void setUp() throws Exception {
        super.setUp();
        /**@todo Bir banka Hesabi Nesnesi olusturuluyor*/
        String hesapNo = "123456789";
        String müşteriAdiSoyadi = "Penbe Macit";
        bankaHesabi = new BankaHesabi(hesapNo, müşteriAdiSoyadi);
    }

    protected void tearDown() throws Exception {
        /**@todo BankaHesabi Nesnesinin referansi siliniyor */
        bankaHesabi = null;
        super.tearDown();
    }

    public void testParaYatir() {
        int miktar = 100;
        String beklenenSonuc = "Hesabiniza "+miktar+ " YTL yatırılmıştır";
        String gercekSonuc = bankaHesabi.paraYatir(miktar);
        assertEquals("Para yatırma", beklenenSonuc, gercekSonuc);
    }

    public void testBankaHesabi() {
        String hesapNo = "123456789";
        String müşteriAdiSoyadi = "Penbe Macit";

        assertEquals("Hesap Numarasi", hesapNo, bankaHesabi.hesapNo);
        assertEquals("Musteri adi ve soyadi", müşteriAdiSoyadi, bankaHesabi.müşteriAdiSoyadi);
        assertEquals("Bakiye", hesapNo, bankaHesabi.hesapNo);
        assertFalse("Acik mi", bankaHesabi.acik);
    }

    public void testParaCek() {
        int miktar = 10;
        String beklenenSonuc0 = "Hesabinizdan " +miktar+" YTL çekilmiştir";
        String beklenenSonuc1 = "Hesabinizda yeterli miktar bulunmadığı için işleminiz yapılamamaktadır";
        String beklenenSonuc2= "Acik olmayan bir hesaptan para çekemezsiniz";

        bankaHesabi.paraYatir(100);
        String gercekSonuc = bankaHesabi.paraCek(miktar);
        assertEquals("Para Çekme", beklenenSonuc0, gercekSonuc);

        miktar = 100;
        gercekSonuc = bankaHesabi.paraCek(miktar);
        assertEquals("Para Çekme", beklenenSonuc1, gercekSonuc);

        miktar = 90;
        gercekSonuc = bankaHesabi.paraCek(miktar);

        miktar = 100;
        gercekSonuc = bankaHesabi.paraCek(miktar);
        assertEquals("Para Çekme", beklenenSonuc2, gercekSonuc);
    }
}
```