Java ve Eclipse Dersleri

MUSTAFA SERHAT DÜNDAR, <SERHAT AT SERHATDUNDAR DOT COM>,
28/02/2009

1) PROJE OLUŞTURMAK

File / New / Project / Java / Java Project kısmından proje oluşturabilirsiniz.

Projemizin adını SerhatDundar olarak belirledim.

2) PAKET OLUŞTURMAK

Projemizin şimdi classları ekleyebileceğimiz bir pakete ihtiyacı var.

Paket isimleri web adresi formatındadır "org.serhatdundar.paket" şeklinde örneklendirebiliriz.

File / New / Package kısmından paket oluşturabiliriz.

Paket ismine org.Serhatdundar.paket yazalım.

Paket ne işe yarar? Paket gündelik hayatta olduğu gibi bir toplayıcı, düzenleyicidir. Sınıfları paketler içine oluştururuz. Bir paketten, başka bir paketteki veriyi çekebiliriz.

*** Paket isimleri nokta ile başlayamaz ve bitemez.**

Paketimiz oluştuğuna göre şimdi bir sınıf yaratmalıyız.

3) CLASS (SINIF) OLUŞTURMAK

Paketimize sağ tıklayıp File / New / Class seçeriz.

Class adını Kisi olarak seçtim. Bu kisinin adını, yaşını belirteceğiz.

```
package org.Serhatdundar.paket;

public class Kisi {

        //Fields
```

```
private String Isim;

private int Yas;


//constructor


public Kisi() {
        Isim = "serhat";
        Yas = 19;
}
```

Şuanda bir sınıfın en temel 2 kısmını oluşturduk. Bunlar fields yani alanlar ve constructor yani işçiler-işi görenler. Bir sonraki kısımda 3.temel özelliği oluşturacağız.

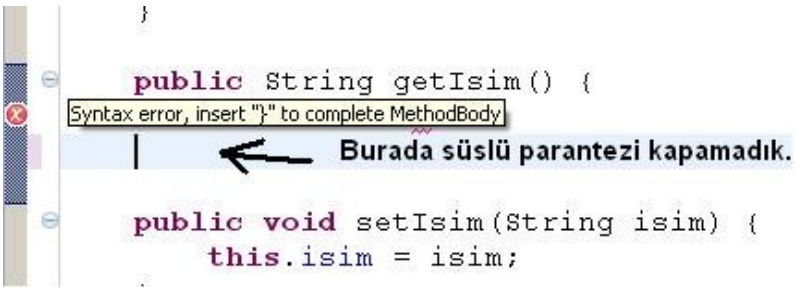
Fields => object'in data elemanlarını barındırır.

Constructor => herhangi bir objeyi belirleyen-isimlendiren kısımdır.

Dikkat ettiyseniz süslü parantezlerimiz içine classlar, metodlar ve diğer kod bloklarını yazıyoruz.

Ve her java cümlesinin sonuna noktalı virgül koyuyoruz.

Bunlardan birini unutursanız Eclipse size bir yerde hata yapıyorsunuz diye uyarır.



Ek olarak eclipse'in mor olarak belittiği kelimeler asla projeniz içinde isim olarak kullanılamaz.

Java büyük ve küçük harfe duyarlıdır. kisi ve Kisi aynı şeyler değildir. Büyük – küçük harf yazılımında hata yaparsanız eclipse sizi uyarır.

Fields'te kullandığımız objeleri constructors'ta tekrar kullanmamız gerektiğinda CTRL+Space tuşları ile eclipse'in bize fields kısmında belirttiğimiz objeleri göstermesini sağlayabiliriz.

4) Getter ve Setter Oluşturmak

Bir sınıf en basit şekilde 3 kısımdan oluşur. Fields Constructors ve Metods.

İlk 2 temel bileşeni oluşturmuştuk. Şuanda 3. temel bileşeni oluşturacağız.

Getter : Sınıf alanlarındaki değerlere dönecek olan metodlar.

Setter : Sınıf alanlarındaki değerleri belirleyecek olan metodlar.

Normalde bu işi manuel olarakta yapabiliriz fakat eclipse bunu bizim yerimize yapacak.

Source / Generate Getters and Setters'e tıkladığınızda otomatik olarak getter ve setter'larınız oluşacak.

Kodumuza eklenen kısımlar neler bi bakalım :

```
public String getIsim() {
        return Isim;
}

public void setIsim(String Isim) {
        this.Isim = Isim;
}

public int getYas() {
        return Yas;
}

public void setYas(int Yas) {
        this.Yas = Yas;
}
```

Bunların tam olarak kullanımlarını deneme-yanılma yöntemiyle bir sonraki başlıkta, Scrapbook ile göreceğiz.

5) SCRAPBOOK KULLANIMI

* Yeni classları test etmek ve komutları denemek için harika.

* Otomatik test imkanı sunmuyor. (Bunun için JUnit Test Case kullanacağız ileriki bölümlerde)

File / New / Java / Java Run-Debug / Scrapbook Page

Scrapbook sayfasına “TestSayfamiz” ismini verdim.

Expression = tek değerden ibaret kod parçasıdır (2+2) – inspect (incele) özelliği ile sonuçlarını görüntüleyebiliriz.

statement = noktalı virgül ile biten tam anlamda kod cümlesidir. (a=3;) execute özelliğini kullanırız.

*** Expression ve Statement kelimelerin anlamlarına bakarsanız ikisininde “ifade” olduğunu görürsünüz. Fakat aradaki fark statement daha kesin bir ifade özelliği taşır.**

Şimdi oluşturduğumuz Scrapbook sayfasında birkaç deneme yapalım.

Örnek 1 :

```
int a = 5;

int b = a * 10;

b
```

Int yani integer yani sayı demektir. A sayısı 5’tir dedik. B sayısı; a sayısı ile 10’un çarpımıdır dedik. B yazarak istediğimiz şeyin b olduğunu belirttik. Kodu highlight ediyoruz yani seçiyoruz. (Üstünü mavi yapmak :)

Kodu seçip inspect ettiğinizde açılan pencerede “50” sonucunu göreceksiniz.

Örnek 2 :

```
int a = 5;

a = a * 10;

a
```

Kodu seçip inspect ettiğinizde açılan pencerede “50” sonucunu göreceksiniz.

Yukarıdaki işlemle aynı sonucu alacaksınız yani 50. İnteger olan a’yı 5e eşitledik. Başka bir a’yı ise integer olan a’nın 10 ile çarpılmasıdır diye ifade ettik. Sonuç olarak a’yı istedik. Bu örneği yazmamın sebebi integer olan a ile a’nın farklı şeyler olduğunu göstermekti.

Örnek 3 :

```
System.out.println("deneme"); // statement yani tam bir kod cümlesi
```

Kodu seçip Execute ettiğinizde Consol’da “deneme” yazısını göreceksiniz.

6) Scrapbook’a Paket İmport Etmek

```
package org.Serhatdundar.paket;
```

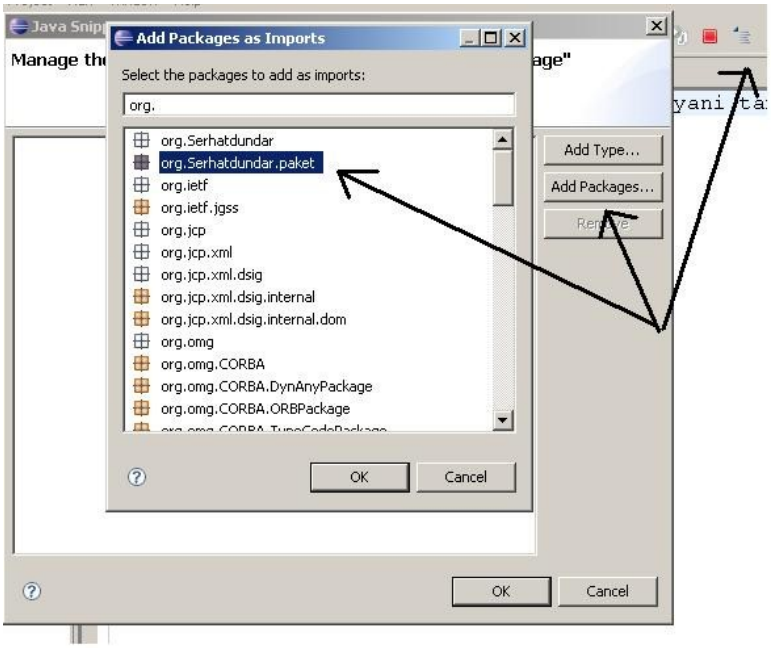
Eclipse sınıfları bunun sonuna . (nokta) işareti ile sınıfları ekleyerek ifade eder.

Örnek :

package org.Serhatdundar.paket.Kisi

Başka bir pakete ait sınıfa ihtiyaç duyarsanız import etmeniz yeter.

Şuan scrapbook sayfamız “Serhatdundar” paketini tanımıyor. Yani ona “Kisi” değişkenini sorsak bilemez. Çünkü bu değişken tanımadığı bir paket içinde. Tanıtmak için Java Snipper İmport kısmından (Set The Import Declarations For Running Code) (stopun yanındaki) add packages diyoruz. “org.Serhatdundar.paket” i seçiyoruz.



Şimdi scrapbook’a Kisi dersem anlayacak ki “org.Serhatdundar.paket” de var bu.

Kisi p = new Kisi ();

p.

Yazdık bisürü metod gördük. Yani neymiş, bütün java metod'ları object class'ının altında barınırmış. Bütün java sınıfları metodlarını object sınıfından miras alırmış.

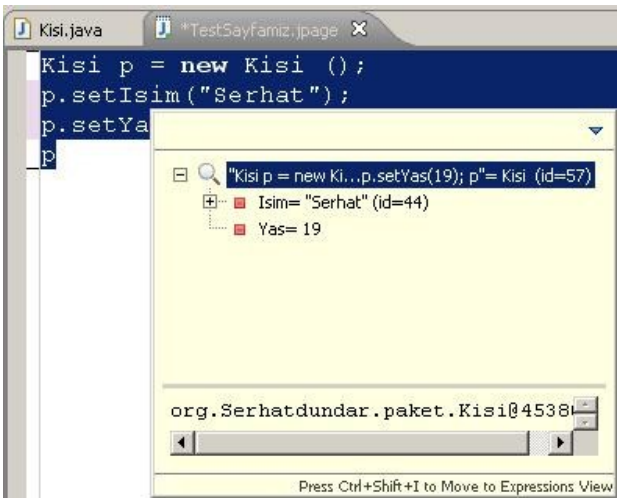
p.<method Isim> = object p'nin üstünde bu methodu çalıştırır.

Eğer metodlar görünmez ise CTRL+Space ile gösterebilirsiniz.

SET METHODU :

```
Kisi p = new Kisi ();
p.setIsim("Serhat");
p.setYas(19);
p
```

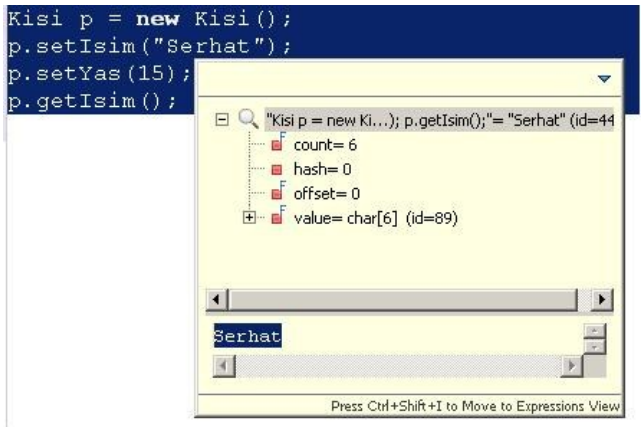
inspect edelim :



GET METHODU :

```
Kisi p = new Kisi();
p.setIsim("Serhat");
p.setYas(19);
p.getIsim();
```

yazdık inspect ettik. Koddan anlayacağınız üzere, adı Serhat, yaşı 19 olsun dedik ve bide bize ismini gösteriver dedik :



Yine aynı kodu highlight edip inspect etmek yerine “Display results of evulating text” seçeneği ile inceleyelim. (Inspect tuşunun yanındadır, eğer yoksa “Run” menüsünden bulabilirsiniz.

Eclipse Çıktısı :

```
Kisi p = new Kisi();

p.setIsim("Serhat");

p.setYas(19);

p.getIsim();

(java.lang.String) Serhat
```

Şimdi de hatalı kod yazmayı deneyelim :

```
Kisi p = new Kisi();

p.setIsim("Serhat); // Çift tırnak ile kapamayı unuttuk.

p.setYas(19);

p.getIsim; // Parantezleri unuttuk.
```

inspect dediğimizde

String literal is not properly closed by a double-quote

Syntax error, insert ")" to complete Expression

uyarısını aldık.

İlk hata kodunda double-quote yani çift tırnak ile kapamayı unuttunuz diyor.

İkinci hata kodunda ise anlayacağınız üzere parantez hatası yaptınız diyor.

7) Unit Testing

* Bütün metodları otomatik test eder.

* Kodda değişiklik yaparken işimize çok yarayacak.

* Güçlü projelerin vazgeçilmezi.

* Eclipse'in içinde hazır gelir.

Test için yeni bir kaynak klasör oluşturmalıyız. Projeyi seçtikten sonra File/New/Source Folder yolundan klasörümüzü oluşturalım. Adını test koyalım.

Test klasörüne yeni paket oluşturalım. Onunda adı org.Serhatdundar.paket olması gerekmekte. Yani test edeceğimiz uygulamanın paketi ile aynı. (Page/New/package)

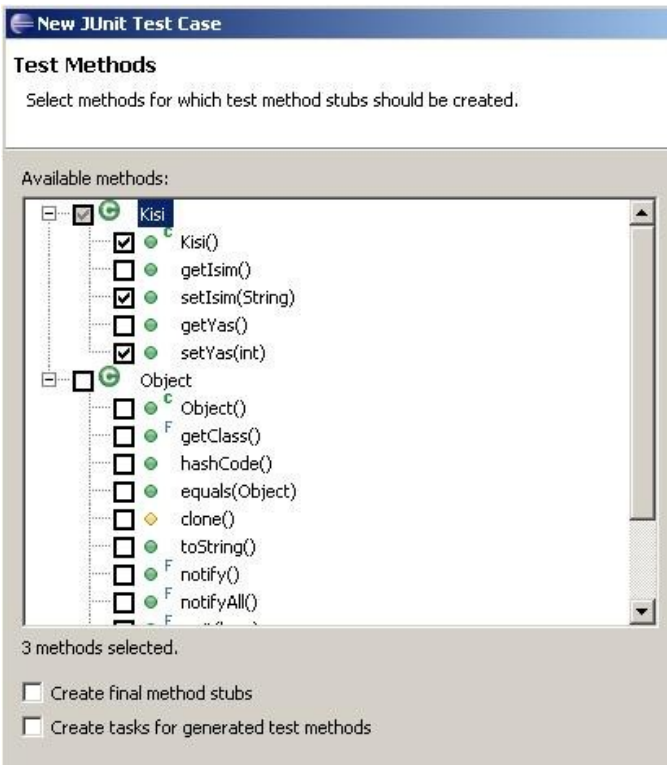
Oluşan pakete sağ tıklayıp. “New / Junit test case” yolundan yeni bir Test Case oluşturun. Name kısmına KisiTest yazdık (siz farklı bişeyde yazabilirsinizi tabiki) “Class under test” kısmına “Kisi” yazdık. Çünkü biz “Kisi” isimli sınıfı test edeceğiz.

Üst kısımda “Superclass does not exist” uyarısı aldık. Bu neden kaynaklandı? Çünkü JUnit3, SerhatDundar projemizin yapılandırma yoluna dahil değil. (Build path=Yapılandırma yolu)

Build Path : Uygulamayı derlemek için gereken harici Java sınıflarının yeri.

JUnit’te yukarıda bahsettiğimiz harici Java sınıflarından biri olduğu için build path’e eklememiz gerekmekte.

Next’e tıklayarak ilerliyoruz.



“Kisi” sınıfı altında kullandığımız tüm metodları karşımızda görmemiz gerekiyor.

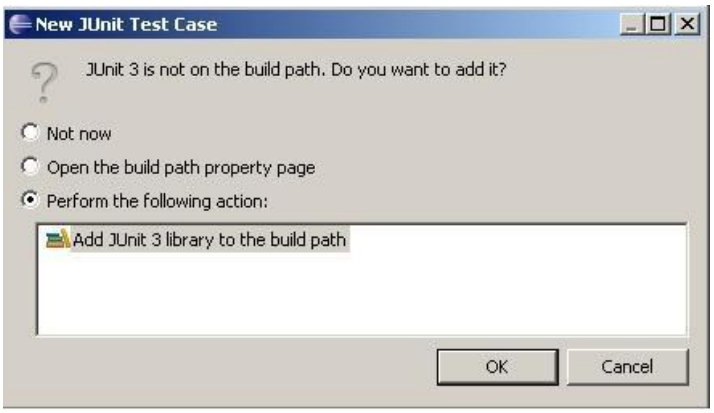
“Object” sınıfı altında ise object sınıfına bağlı superclass’ları görebilmekteyiz.

Eclipse bu ekranda seçtiğimiz metodlar için içi boş test metodları oluşturacak.

Biz projemizde “Kisi()” , “setIsim(String)”, “setYas(int)” metodlarını test edeceğiz. Bunları seçelim.

Finish’e tıklayalım.

- Yukarıda belirttiğimiz alt kısımdaki uyarı mesajı görünmezse panik yapmayın, Finish’e tıkladığımızda bize JUnit Test Case’i build path’e ekleyeyim mi diye sorulacak :



OK'e basarak geçelim.

```
package org.Serhatdundar.paket;

import junit.framework.TestCase;

public class KisiTest extends TestCase {

	public void testKisi() {

		fail("Not yet implemented");

	}

	public void testSetIsim() {

		fail("Not yet implemented");

	}

	public void testSetYas() {

		fail("Not yet implemented");

	}

}
```

*** Şuan eclipse'in projemizin altına JUnit3 isminde bir kütüphane oluşturması gerekiyor. Eğer bu oluşmadıysa sayfanızda bir çok hata göreceksiniz. Bu hataları düzeltmek için "Problems" bölümüne gelip ilgili probleme sağ tıklayıp "Quick Fix"i seçebilirsiniz.

Quick Fix Nedir?

*Derleyici hatalarını düzeltmek için bize önerilerde bulunur.

*Hataya sağ tıklayarak veya CTRL+1 kombinasyonu ile ulaşılabilir.

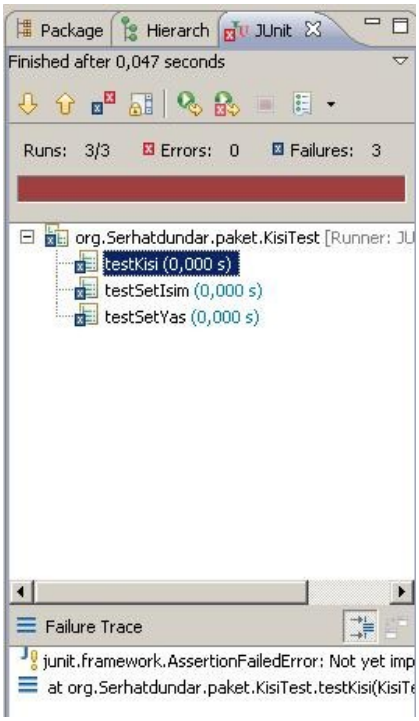
*Programcıların işini kolaylaştıran pratik bir özelliktir.

Quick Fix'in bizim yerimize yaptığı işi görmek için SerhatDundar'a sağ tıklayın. Build Path kısmından Configure Build Path'i seçin. Üst menüden Libraries'i seçerseniz JUnit3 kütüphanesini görebilirsiniz. Biz bu işi Add Library kısmından bizde yapabilirdik fakat eclipse bunu bizim yerimize kolay bir şekilde yaptı.

Fail("Not yet implemented") uyarısı bize TestCase'in henüz görevini yapmamış olduğunu bildiriyor.

Çalışmasını kontrol etmek için Run/Run As/Junit Test yolunu seçiniz.

Test sonucunda 3 adet failure yani eksiklik tespit edilmesi gerekiyor. Bu beklediğimiz bir sonuç.



Failure Trace kısmından neden hata aldığımızı görebilirsiniz.

Oluşan kod içinden "fail("Not yet implemented");" satırlarını silelim.

İlk sildiğimiz satırda boşalan yere yeni bir Kisi nesnesi oluşturup bu alanın doğru yapılanıp yapılanmadığını test etmeliyiz.

```
package org.Serhatdundar.paket;

import junit.framework.TestCase;

public class KisiTest extends TestCase {

	public void testKisi() {

		fail("Not yet implemented");

	}

	public void testSetIsim() {

		fail("Not yet implemented");
```

```
	}
	public void testSetYas() {
		fail("Not yet implemented");
	}
}
```

Şimdi TestCase sınıfının bir nesnesi olan assertequals'i inceleyelim.

Assertequals'in temel mantığı :

assertEquals("Beklenen Değer", "gelen değer");

şeklindedir. Bunu örnek üzerinde görelim :

```
package org.Serhatdundar.paket;
import junit.framework.TestCase;
public class KisiTest extends TestCase {
	public void testKisi() {
	Kisi k1 = new Kisi();
	assertEquals("serhat", k1.getIsim());
	assertEquals(19, k1.getYas());
		}
	public void testSetIsim() {
	Kisi k2 = new Kisi();
	k2.setIsim("Serhat");
	assertEquals("Serhat", k2.getIsim());
	}
	public void testSetYas() {
	Kisi k3 = new Kisi();
	k3.setYas(19);
	assertEquals(19, k3.getYas());
	}
}
```

* assertEquals statik yani sabit bir metod'dur.

STATİK METOD :

- Tamamen bir sınıfa bağlıdır.
- Sınıfın herhangi bir aşamasına bağlı değildir.
- Syntax yani sözdizimi <class>.<method> şeklindedir. Örneğin KisiTest.assertEquals(…)
- Eğer aynı sınıf içinde belirtiliyorsa <class> kısmını atlayabiliriz. Bu ne anlama geliyor detaylı görelim:

6.satırımız :

```
assertEquals("serhat", k1.getIsim());
```

istersek bunu

```
KisiTest.assertEquals("serhat", k1.getIsim());
```

Şeklinde de yazabiliriz. Fakat başta :

```
public class KisiTest extends TestCase {
```

şekinde belirttiğimiz için ve zaten KisiTest sınıfının içinde olduğumuz için <class adı> kısmını geçebiliriz.

8) TEST-DRİVEN DEVELOPMENT (TDD)

- Gelişmiş, üst düzey yazılımların anahtar bileşenlerinden biridir.
- Test edilecek metodu yazmadan önce UnitTest metodunu kullanırız.
- Unit Test'ler kodumuz için belirleyici ve dökümentasyon olarak olarak kullanılır.
- eClipse'nin harika bir TDD desteği vardır.

8.1) TOSTRİNG METODU

- Temeli object (nesne) sınıfıdır.
- Nesneye bir string görevi döndürür.
- Normalde her sınıf için oluşturulur.

toString metodunun ne iş yaptığını anlamak için deneme tahtamız olan Scrapbook'u kullanabiliriz.

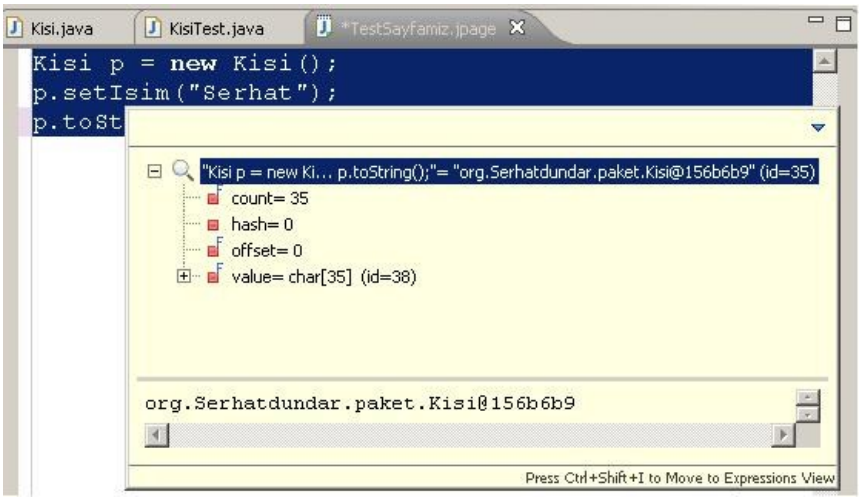
Scrapbook sayfası yaratıp

```
Kisi p = new Kisi();
```

p.setIsım(“Serhat”);

yazarak Kisi sınıfını tanımlayalım ve Serhat değerini atayalım.

Şimdi bu koda p.toString metodunu’da ekleyerek ne olacağını görelim.



İlk satırda gördüğünüz “org.Serhatdundar.paket.Kisi” bizim sınıfımızın tam yolu. Sonra gelen @156b6b9 kısmı ise hex kısmı.

Herhangi bir nesne icinde bu metodu yazarsaniz nesnenin iceriginin size nasil gorunmesine iliskin bir kod yazmaniz beklenir. Mesela

```
public class StringDeneme
{
   private String isim = "Serhat";
   private String soyisim = "Dundar";

   public String toString()
   {
      return "Adi:" + isim + " Soyadi:" + soyisim;
   }

   public static void main(String[] args)
   {
```

```
        StringDeneme test = new StringDeneme();
        System.out.println("to String sonucu: " + test.toString());
    }
}
```

Sonucta ekranda "to String sonucu: Adi:Serhat Soyadi:Dundar" yazdigini goreceksiniz.

8.2) Overriding a Method

Yazmış olduğumuz bir metodu süregelen başka bir metoda dönüştürürken overriding kullanırız.

8.3) Test-Driven Development

- Metodun hangi işi gördüğünü düşünün.
- Metodu test eden bir test case hazırlayın.
- Yeni metodu hazırlayın.
- Yeni metodu test edin.
- Sonuç olumluysa başarmışsınız demektir.

Atacağımız adımları belirlediğimize göre kodumuzu hazırlamaya başlayalım :

KisiTest.Java:

```
package org.Serhatdundar.paket;
import junit.framework.TestCase;
public class KisiTest extends TestCase {
        public void testKisi() {
        Kisi k1 = new Kisi();
        assertEquals("serhat", k1.getIsim());
        assertEquals(19, k1.getYas());
                }
```

```
	public void testSetIsim() {
	Kisi k2 = new Kisi();
	k2.setIsim("Serhat");
	assertEquals("Serhat", k2.getIsim());
	}
	public void testSetYas() {
	Kisi k3 = new Kisi();
	k3.setYas(19);
	assertEquals(19, k3.getYas());
	}
	public void testToString(){
	Kisi k4 = new Kisi();
	k4.setIsim("Serhat Dundar");
	k4.setYas(19);
	}
}
```

Peki şuanda toString metodundan ne yapmasını istedik?

Kişilerin ismini ve yaşını parantez içinden getirmesini istedik :

```
package org.Serhatdundar.paket;
import junit.framework.TestCase;
public class KisiTest extends TestCase {
	public void testKisi() {
	Kisi k1 = new Kisi();
	assertEquals("serhat", k1.getIsim());
	assertEquals(19, k1.getYas());
		}
	public void testSetIsim() {
	Kisi k2 = new Kisi();
	k2.setIsim("Serhat");
```

```
        assertEquals("Serhat", k2.getIsim());
        }
        public void testSetYas() {
        Kisi k3 = new Kisi();
        k3.setYas(19);
        assertEquals(19, k3.getYas());
        }
        public void testToString(){
        Kisi k4 = new Kisi();
        k4.setIsim("Serhat Dundar");
        k4.setYas(19);
        String testString = "Serhat Dundar (19 yasinda)";
        assertEquals(testString, k4.toString());
        }
}
```

Burada toString() metodunda yapılacak herhangi bir değişiklik testin hata vermesine yol açacaktır. Öyleyse biz bu ifadenin, toString() metodu testten geçene kadar tamamen doğru olması gerektiğini biliyoruz.

Şimdi bu yazdığımız metodu JUnit Test olarak çalıştıralım.

(Run/Run As/JUnit Test & kodlar seçilmiş olmalı)

4 metodumuzdan testToString hata verdi. Çünkü hala metodumuzu tam olarak yazmadık.

Şimdi test paketimizden çıkıp Kisi.Java dosyamıza gidelim.

Kisi.Java dosyamıza aşağıdaki kodları eklememiz gerekmekte.

```
        public String toString(){
```

```
        return this.getIsim() + "(" + this.getYas() +          "yas)";
    }
```

Bu kodun anlamı :

İsim bilgisini getir.

(+) işareti bağlaç görevi görmekte.

Parantez aç.

Yas bilgisi getir.

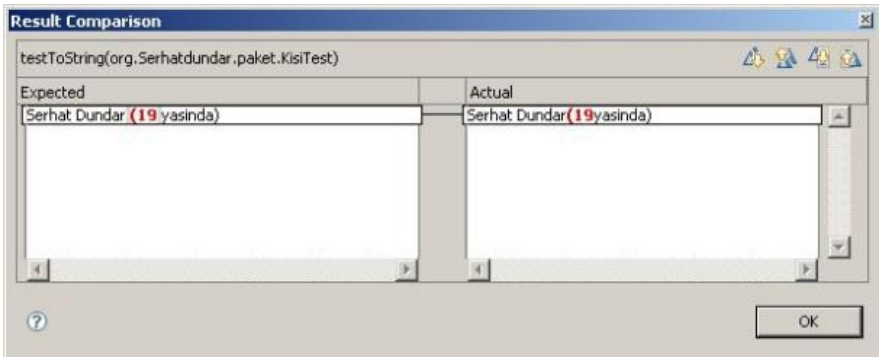
Yas yaz ve parantezi kapat.

Bu kodu yazarken this.getIsim yerine this.Isim'de yazsak sonuç aynı olurdu. Fakat this.getIsim şeklinde belirtmemiz bize ileride bazı kolaylıklar sağlayacak. Buna bir örnek vermek gerekirse; Isim alanını iki parçaya bölmek istediğimizde sadece return firstName + "" + lastName şeklinde belirtmemiz yetecek.

Şimdi tekrar JUnit testimizi çalıştıralım.

Opps. Hala toString metodunda bir hata var. Expected test results bölümünden sorunun ne olduğunu görelim.

Aldığımız hatanın tek sebebi sadece 1 tane boşluk karakteri.



Kisi.Javaya geri dönüp gereken boşlukları ekleyelim.

```
    public String toString() {
        return this.getIsim() + " (" + this.getYas() +          " yasinda)";
    }
```

Tekrar JUnit testing yaptığınızda hiçbir sorun olmadığını göreceksiniz.

Şuana kadar ilk önce Kisi sınıfını oluşturmuştuk, daha sonrasinda KisiTest sınıfını oluşturduk. Bu bölümde yapacağımız iş biraz daha farklı. Kitap sınıfını oluşturmadan önce KitapTest sınıfını oluşturacağız.

İlk önce KitapTest için bir JUnit Test Case oluşturmalıyız.

Serhatdundar projesi altındaki test klasörüne ulaşalım. Org.Serhatdundar.paket ismindeki paketimize sağ tıklayıp daha önce öğrenmiş olduğumuz şekilde bir JUnit oluşturalım.

İsmini KitapTest olarak belirledim. Fakat bu sefer “Class under test” kısmını boş bırakacağız. Çünkü henüz sınıfımızı oluşturmadık. Başta belirttiğim gibi önce test sınıfını oluşturuyoruz.

Şimdi Kitap sınıfımızın ne iş göreceğini kararlaştıralım.

Bu sınıfımız Baslik, Yazar ve Kisi kısımlarından oluşsun.

Baslik kısmı kitabın ismi, Yazar kısmı kitabın yazarı, Kisi kısmı ise kitabı alan kisinin ismini tutacak.

KitapTest.Java dosyamızı oluşturmaya başlayalım :

```
package org.Serhatdundar.paket;

import junit.framework.TestCase;

public class KitapTest extends TestCase {

	public void testKitap() {

		Kitap k1 = new Kitap("Basics of programming");

		assertEquals("Basics of programming", k1.baslik);

		assertEquals("Mark Dexter", k1.yazar);

	}

}
```

k1 olarak tanımlı kitabımızın adının “Basics of programming” olmasını istedik.

Bir alt satırda ise Beklenen ve Gelen değerleri karşılaştırıyoruz. (AssertEquals)

Beklenen değer Basics of programming, gelen değer k1.baslik.

Şuan için k1.baslik herhangi bir sonuç vermeyecektir. Çünkü henüz Kitap sınfını ve başlığı tanımlamadık. Yazarın beklenen değeri Mark Dexter, gelen değer ise k1.yazar. Yine k1.yazar şuan Java için bilinmeyen bir sonuç.

Şu ana kadar hazırladığımız kısmı Test edelim (Run/Run as/JUnit Test)

Aşağıdaki şekilde bir hata almamız normal. Çünkü şuan işleri tersten götürüyoruz.

Failure Trace kısmından göreceğiniz gibi eclipse için şuan da Kitap bilinmeyen bir tip.

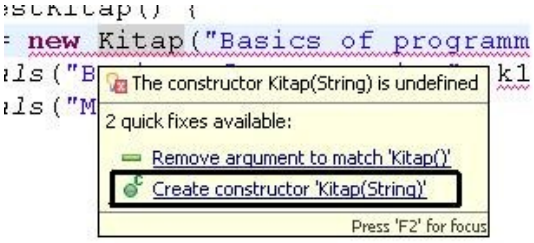
Öyleyse bu sorunu eclipse'nin nimetlerinden faydalanarak çözelim.

"Kitap bilinmeyen bir tip" uyarısının altını çizdim. Eclipse bize bu sorunu çözmek için 9 tane seçenek sunuyor. Biz eclipse'den Kitap sınıfını oluşturmasını isteyeceğiz. Eclipse bunu yaparak bizi bir çok zahmetten kurtaracak. "Create class Kitap" a tıklayalım.

Eclipse bu durumda otomatik olarak Source Folder kısmında SerhatDundar/Test seçmiş olacak. Biz bu sınıfı test klasörü altında değil src klasörü altında oluşturmak istediğimiz için bu kısmı src ile değiştiriyoruz.

Finish'e tıkladığımızda karşımızda boş bir Kitap sınıfı görmekteyiz.

Şimdi tekrar KitapTest'e dönerek diğer sorunları çözelim.



Diğer hataya tıkladığımızda 2 çözüm alternatifimiz var. İlki bu kod parçasını kaldırır. İkincisi yani bizim seçeceğimiz ise Kitap metodunu oluşturur.

Kisi.Java altında şuanda otomatik olarak kalıp metodumuz oluştu.

Yeniden KitapTest.Java'ya dönelim. Bir diğer hata veren bölüm olan k1.baslik kısmına tıkladığımızda yine 2 otomatik çözüm göreceğiz.

Create Field yani "Alanı Oluştur" bizim işimizi görecek olan çözüm.

Otomatik olarak "Baslik" alanımızda oluşuyor.

K1.yazar hatamızıda aynı yöntemle düzelttirelim.

Kodumuzun içine eclipse'nin oluşturduğu yorum kısımlarını eclipse içinde Window/Show View/Show Tasks bölümünden görebilirsiniz. Bu yorum satırları bu bölüm altında Task (Görev) olarak kaydedilmektedir.

Şimdi metod kısmını tanımlayalım.

```
package org.Serhatdundar.paket;

public class Kitap {

	public String baslik;

	public String yazar;


	public Kitap(String string) {

		this.baslik = string;

		this.yazar = "Mark Dexter";


	}

}
```

this.'den sonra CTRL+SPACE tuş kombinasyonuyla eclipse'nin pratik özelliklerinden faydalanabilirsiniz.

Bu yaptığımız değişikliklerden sonra tekrar KisiTest.Java'yı teste sokalım.

Gördüğünüz gibi hiç hatayla karşılaşmadık.

Yine eclipse'in güzelliklerinden faydalanarak getter ve setter'ları oluşturmasını sağlayacağız.

Source sekmesi altından bunun yapıldığını önceki bölümlerde söylemiştik.

Getbaslik, setbaslik, getyazar metodlarını oluşturalım. Şuanda setyazar'a ihtiyacımız yok.

Eclipse bizim yerimize bu 3 metodu oluşturdu :

```
package org.Serhatdundar.paket;

public class Kitap {
	public String baslik;
	public String yazar;
	public Kitap(String string) {
		this.baslik = string;
		this.yazar = "Mark Dexter";
	}
	public String getBaslik() {
		return baslik;
	}
	public void setBaslik(String baslik) {
		this.baslik = baslik;
	}
	public String getYazar() {
		return yazar;
	}
}
```

Yaptığımız işlemlere bir göz atarsak :

- KitapTest sınıfı oluşturuldu.
- testKitap metodu oluşturuldu.
- Eclipse'nin quick fix özelliği ile hatalarımız düzeldi.
- Kitap sınıfı için constructor'lar oluşturuldu.
- Junit test yapıldı ve başarıyla sonuçlandı.

ÖDÜNÇ KİTAP PROJESİ :

- Şuan Kisi ve Kitap sınıflarımız var.
- Hangi kişinin hangi kitabı ödünç aldığını henüz belirtmedik.
- Kisi ve Kitap sınıfları arasında bir bağlantı kurmalıyız.
- Test-First yani önce Test sınıflarını oluşturduğumuz geliştirme yöntemini uygulamaya devam edeceğiz.

İlk olarak KitapTest sınıfımızı açalım ve Kisi'yi test eden metodumuzu hazırlayalım.

Yapacağımız işlemlerin algoritması kısaca :

- testGetKisi() metodu hazırlayacağız.
- getKisi() metodu hazırlayacağız.
- Test'i çalıştıracağız.

KitapTest.Java :

```
public void testGetKisi() {
		Kitap b2 = new Kitap("Savas ve Baris");
		Kisi k2 = new Kisi();
		k2.setIsim("Serhat");
		// Bu metod, belirtilen kitabın bu kisiye kiralandıgını belirtiyor.
		b2.setKisi(k2);
		// Kitaba sahip olan kisiyi belirtecegiz.
		Kisi testKisi = b2.getKisi();
		String testIsim = testKisi.getIsim();
		assertEquals("Serhat", testIsim);
	}
```

Daha önceki derste öğrendiğimiz gibi Quick Fix yardımı ile hataları düzelttirelim.

Otomatik olarak Kitap.Java içinde aşağıdaki satırlar oluşacak :

```
public void setKisi(Kisi k2) {
		// TODO Auto-generated method stub
	}
```

```
public Kisi getKisi() {
        // TODO Auto-generated method stub
        return null;
}
```

Şimdi Kitap.Java içindeki yeni eklenen set ve get metodlarını düzenleyelim.

```
this.kisi = k2;
```

Şuan “kisi” sözcüğü yine hata vermekte. Bunu da quick fix yardımı ile düzeltelim.

Kitap.Java içinde

```
private Kisi kisi;
```

satırının oluştuğunu göreceksiniz.

Burada Sınıf adının büyük harfle, alan adının küçük harfle başladığına dikkat ediniz.

8 Ders’tir kodlarımızı public ve private olarak başlatıyoruz. Peki bunlar ne anlama gelir ve nedir?

Public ve private; “Access modifiers” olarak tanımlanır. Türkçe anlamı ile “erişim cümlecikleri”.

Diğer sınıfların bu metod veya alanları kullanabilme izinlerinin tanımlanmasını sağlayan cümleciklerdir.

Java’da 4 çeşit “erişim cümleciği” mevcuttur.

Public : Herhangi bir sınıftan erişime açık.

Private : Sadece bu sınıftan erişilebilir.

Herhangi bir erişim cümleciği olmadığı durum : Sadece bu paketten erişilebilir.

Eğer kodumuzda ki “erişim cümlecikleri”ni silersek erişimi sadece aynı paket içiyle sınıflandırmış oluruz.

Şimdi erişim cümleciklerini silerek KitapTest.Java’nın Kitap.Java’da mevcut alanlara ve metodlara erişmesini, onu test edebilmesini sağlayacağız.

```
String baslik;
String yazar;
Kisi kisi;
```

(Erişim cümlecikleri silinmiş hal)

Şimdi getKisi metodunu hazırlayalım. Bu oldukça basit bir işlem.

```
public Kisi getKisi() {
        return this.kisi;
}
```

Şimdi KitapTest.Java'ya giderek JUnit test'i çalıştıralım.

Gördüğümüz gibi hatasız biçimde çalıştı.

Şimdi Kitap ve Kisi sınıfları arasında ilişki kuralım :

- Kitap ve Kisi sınıfları iliskili.
- Kitap sınıfı Kisi sınıfına bağlı.
- 1'e 1 ilişki söz konusu (1 kitabın belirli bir zaman diliminde 1 sahibi olabilir, 2 kişi aynı kitabı aynı anda okuyamaz.)

Kitap sınıfının içinde "kisi" adını andığımız yerlere bakalım:

```
public void setKisi(Kisi k2) {
        this.kisi = k2;
    }
    public Kisi getKisi() {
        return this.kisi;
    }
```

Daha önce KitapTest.Java'da yazdığımız şu satırlara dönelim :

```
Kisi testKisi = b2.getKisi();
String testIsim = testKisi.getIsim();
```

Burada aslında kullandığımız 2 satırlık kod gereksiz. Bunu tek satır halinde yazabilme imkanımız var.

- Eclipse'nin kullanışlı özelliklerinden biri highlight ettiğimiz satırın başına yorum işaretlerini otomatik olarak atayabilmesidir. "CTRL (+) /" kombinasyonu ile bu işlemi gerçekleştirebilirsiniz.
- Aynı tuş kombinasyonu ile yorum işaretleri kaldırabilirsinizde.
- Tuş kombinasyonunu "CTRL (+) SHIFT (+) 7" şeklinde de ifade edebiliriz.

Şimdi bu iki satırlık kodumuzu yukarıda belirttiğim şekilde yorum haline çevirip bunu tek satır halinde yazmaya çalışalım.

```
String testIsim = b2.getKisi().getIsim();
```

Burada dikkat ederseniz 2 metod arasında nokta kullarak onları bağladık.

B2.GetKisi() metodunu bir kisiyi cekmek icin kullandık ve sonrasında ki kısımda Kisi nesnesini kullanarak getIsım() metodunu çalıştırdık.

```
//        Kisi testKisi = b2.getKisi();
//        String testIsim = testKisi.getIsim();

          String testIsim = b2.getKisi().getIsim();

          assertEquals("Serhat", testIsim);
```

Şimdi hatalı bir şey yapıp yapmadığımızı anlamak için testimizi tekrar çalıştıralım.

Göreceğiniz gibi herhangi bir hata oluşmayacak.

Şuan 2 tane test sınıfımız var ve şimdi birkaç tane daha oluşturacağız.

Test klasörü altındaki org.Serhatdundar.paket ismindeki paketimizi seçelim (highlight). Sağ tıklayıp, New, Other sekmelerini seçelim.

Junit altındaki, JUnit Test Suite'i seçelim.

Default olarak bizim yazdığımız 2 test sınıfı seçili olması gerekiyor. Böyle değilse seçelim, öyle ise Fisih diyerek işlemi tamamlayalım.

AllTest ismindeki sınıfımızın oluştuğunu göreceksiniz.

AllTest.Java :

```
package org.Serhatdundar.paket;

import junit.framework.Test;

import junit.framework.TestSuite;

public class AllTests {

	public static Test suite() {

		TestSuite suite = new TestSuite("Test for org.Serhatdundar.paket");

		//$JUnit-BEGIN$

		suite.addTestSuite(KisiTest.class);

		suite.addTestSuite(KitapTest.class);

		//$JUnit-END$

		return suite;

	}

}
```

Burada görmüş olduğunuz

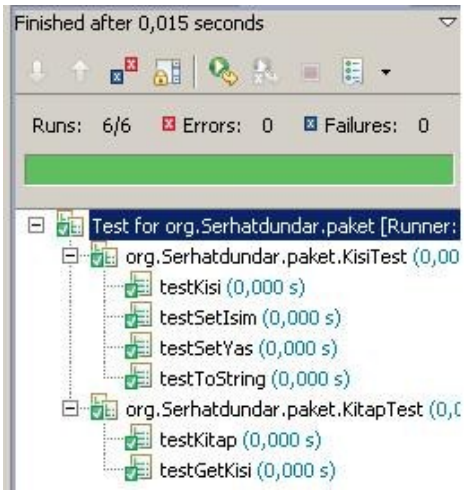
```
suite.addTestSuite(KisiTest.class);

suite.addTestSuite(KitapTest.class);
```

satırları bu suite'e bizim test sınıflarımızın nasıl eklendiğini gösteriyor. Bunu bilmenize gerek yok fakat ileride yni bir test sınıfı oluşturursanız manual olarak buraya o sınıfları yeni bir satır içinde ekleyebilirsiniz.

Şimdi AllTest.Java'yı Junit Testine sokalım.

Gördüğünüz gibi yine bütün testler başarılı.



JUnit Test Suite'in faydası; bütün test sınıflarını tek tek Junit test'e sokmanız gerekmez. Bir JUnit Test Suite oluşturup bütün test sınıflarını buna dahil ederek ve sadece bu suite'i çalıştırarak işlerinizi kolaylaştırabilirsiniz.

M.SERHAT DÜNDAR