Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

Ders İşleyiş Bilgilendirmesi

- Teorik Ders Saatleri: **Çarşamba 14:00 - 16:50**
- Dersler şimdilik online ortamda olacak.
- Duyurular için: https://avesis.yildiz.edu.tr/fcakmak/dokumanlar
  - OBS Toplu Mesaj Sistemi
- Google Classroom Sınıf Kodu: **txgc2fe**
  - Dersle ilgili sorularınızı Classroom üzerinden duvar paylaşımı yaparak sorabilirsiniz.
- Zoom Personal Room:
  - https://us04web.zoom.us/j/3752287039?pwd=TTFKNittZWJTUEhHREovckl0VTVYUT09
  - Meeting ID: 375 228 7039
  - Passcode: 5AXMNd
- Dersle ilgili soru ve danışmalarınız için EN AZ 1 (BİR) GÜN ÖNCEDEN linkten randevu alınız.
  - https://fcakmak.simplybook.it/v2/
  - Mail ile randevu verilmeyecektir.
  - Alınan randevu saatinde eğer randevunuz online ise lütfen yukarıda verilen Zoom odasında, yüzyüze ise D-122 nolu odada bulununuz.

Ders Tanıtım Formu ve Konular

Hafta	Tarih	Konular
1	01.03.2023	Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavramları
2	08.03.2023	UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması
3	15.03.2023	Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları
4	22.03.2023	Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları
5	29.03.2023	Toplama ve Meydana Gelme İlişkileri
6	05.04.2023	Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı
7	12.04.2023	Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi)
8	19.04.2023	Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları)
9	26.04.2023	1. Ara Sınav
10	03.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar
11	10.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar (Devam)
12	17.05.2023	Paralel Programlamaya Giriş
13	24.05.2023	2. Ara Sınav
14	31.05.2023	Grafik User Interface (GUI) Giriş

Ders Tanıtım Formu – Ders Kitabı

	Kitaplar
1	Java How to Program, Harvey M. Deitel & Paul J. Deitel, Prentice-Hall.
2	Core Java 2 Volume I and II, C. S. Horstmann and G. Cornell, Prentice-Hall.
3	UML Distilled, Martin Fowler, Addison-Wesley

Ders Tanıtım Formu – Değerlendirme

	Yöntem	Adedi	Etki Oranı (%)
Başarı Değerlendirme Sistemi	Ara Sınavlar	2	40
	Kısa Sınavlar	-	-
	Ödevler	-	-
	Projeler	1	20
	Dönem Ödevi	-	-
	Laboratuvar	-	-
	Diğer	-	-
	Final Sınavı	1	40

Ders Tanıtım Formu - Bilgilendirme

- Ödevler, sınavlar ortak olacak
- Sınavda sınıf düzeni için 15 dk. önce gelinmeli
- İmza tükenmez kalem ile olmalı
- Yoklama
  - Teorik ders: %70 (< F0)
  - Laboratuvar dersi: %80 (< F0)
- İmza nedir? Neden atılır?
  - Vekalet yok !
- İletişim: fcakmak@yildiz.edu.tr

JAVA

- JAVA
  - JAVA SE
    - Programlama Metodolojisi Desteği
      - Procedural programming
      - Object-oriented programming
      - Generic programming
      - Functional programming
  - JAVA EE
  - JAVA ME

NYP Kavramları

- Niye NYP?
  - Nesneler tekrar kullanılabilir.
- Sınıf (Class)
- Nesne (Object)
- Metot (Method)
- Üye Alan (Attribute - Behaviors)
- Instance
- Encapsulation
- Kalıtım (Inheritance)
- Arayüzler (Interfaces)
- The UML (Unified Modeling Language)
- JRE
- JDK

Sabırla Dinlediğiniz İçin Teşekkürler

Öğr. Grv. Furkan Çakmak

Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

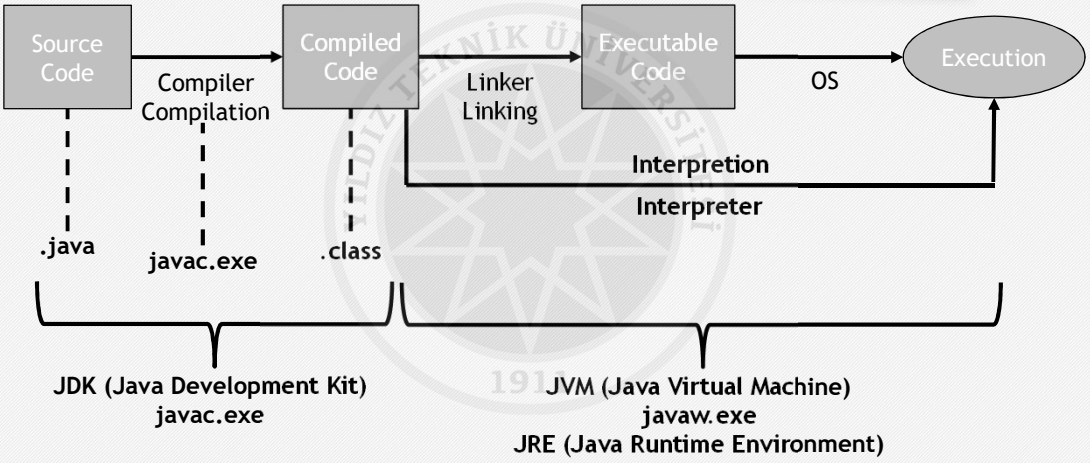
Ders Tanıtım Formu ve Konular

BLM2012 Nesneye Yönelik Programlama Hafta 1

Hafta	Tarih	Konular
1	01.03.2023	Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavramları
2	08.03.2023	UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması
3	15.03.2023	Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları
4	22.03.2023	Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları
5	29.03.2023	Toplama ve Meydana Gelme İlişkileri
6	05.04.2023	Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı
7	12.04.2023	Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi)
8	19.04.2023	Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları)
9	26.04.2023	1. Ara Sınav
10	03.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar
11	10.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar (Devam)
12	17.05.2023	Paralel Programlamaya Giriş
13	24.05.2023	2. Ara Sınav
14	31.05.2023	Grafik User Interface (GUI) Giriş

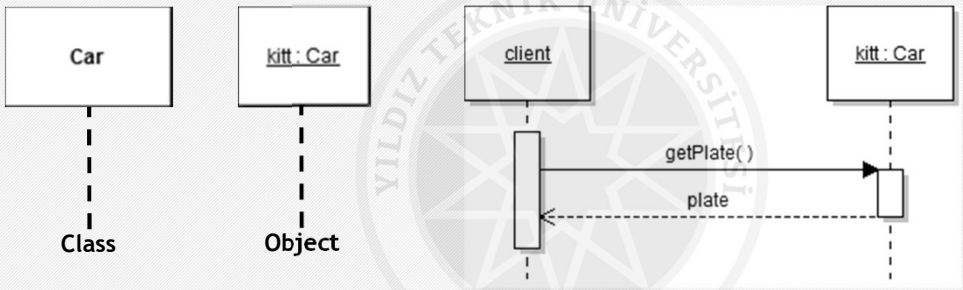
Öğr. Grv. Furkan ÇAKMAK

JAVA EXECUTION ENVIRONMENT



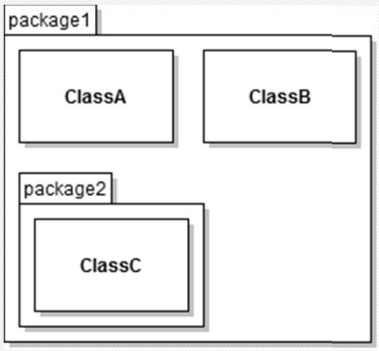
UML Representation (Sequence/Interaction Diagram)

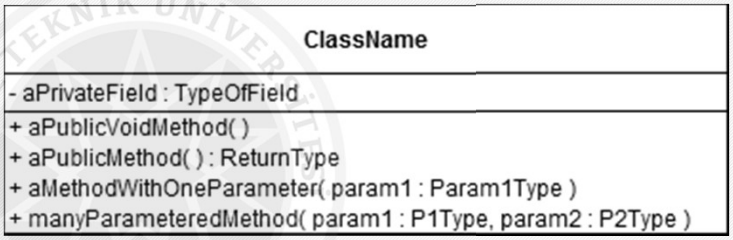
BLM2012
Nesneye
Yönelik
Programlama
Hafta 2



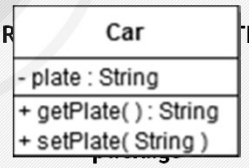
Öğr. Grv. Furkan ÇAKMAK

UML Representation (Devam) (Sequence/Interaction Diagram)





VISIBILITY R ION HIDING

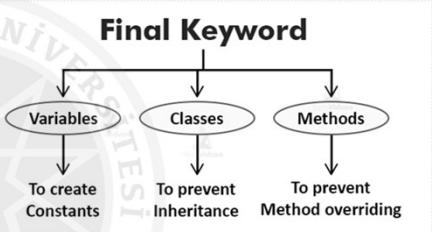


SPECIAL CASES OF MEMBERS

BLM2012 Nesneye Yönelik Programlama Hafta 2

- Static member fields
- Static member methods
- Final variables, Classes, Methods

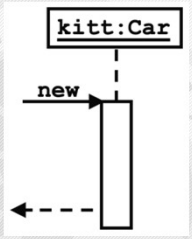
- Final: Only once
- Static: Shared usage



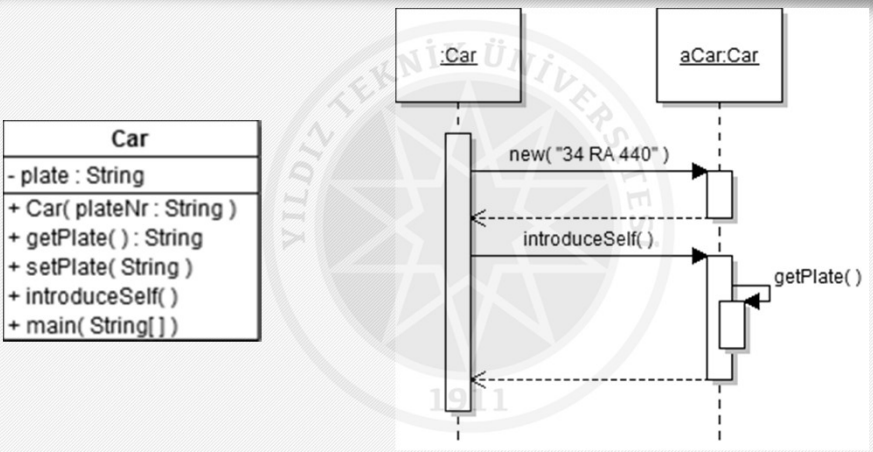
Öğr. Grv. Furkan ÇAKMAK

CONSTRUCTORS AND FINALIZERS

- Constructor Method
  - Overloading
- Finalizing Method
  - object.finalize()



CREATING YOUR OWN CLASS AND OBJECTS



PRIMITIVES AND WRAPPERS

- Wrapper?
  - You can think of a wrapper as a class with only one member field of a primitive type that it wraps/boxes.
  - java.lang

  - int compareTo(Integer anotherInteger)
  - int intValue()
  - static int parseInt(String s)
  - String toString()
  - static String toString(int i)
  - static Integer valueOf(String s)

String Class

- int length()
- int compareTo(String anotherString)
- int compareToIgnoreCase(String str)
- boolean equals(String anotherString)
- boolean equalsIgnoreCase(String anotherString)
- boolean contains(String anotherString)
- String toUpperCase( )
- String toLowerCase( )

Math Class

- random()
- abs(<primitive> a)
- max(<primitive> a, b)
- min(<primitive> a, b)
- ceil(double a)
- floor(double a)
- round(double a)
- sqrt(double a)

More on Random Values (java.util.Random)

- public boolean nextBoolean( ) returns [false, true]
- public double nextDouble( ) returns [0.0, 1.0)
- public float nextFloat ( ) returns [0.0, 1.0)
- public int nextInt( ) returns (-232, 232)
- public int nextInt(int bound) returns [0, bound)
- public long nextLong( ) returns [0, 248)

Sabırla Dinlediğiniz İçin Teşekkürler

Öğr. Grv. Furkan Çakmak

Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

Ders Tanıtım Formu ve Konular

BLM2012 Nesneye Yönelik Programlama Hafta 1

Hafta	Tarih	Konular
1	01.03.2023	Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavramları
2	08.03.2023	UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması
3	15.03.2023	Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları
4	22.03.2023	Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları
5	29.03.2023	Toplama ve Meydana Gelme İlişkileri
6	05.04.2023	Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı
7	12.04.2023	Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi)
8	19.04.2023	Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları)
9	26.04.2023	1. Ara Sınav
10	03.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar
11	10.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar (Devam)
12	17.05.2023	Paralel Programlamaya Giriş
13	24.05.2023	2. Ara Sınav
14	31.05.2023	Grafik User Interface (GUI) Giriş

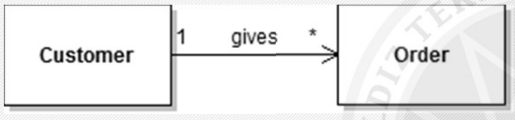
Öğr. Grv. Furkan ÇAKMAK

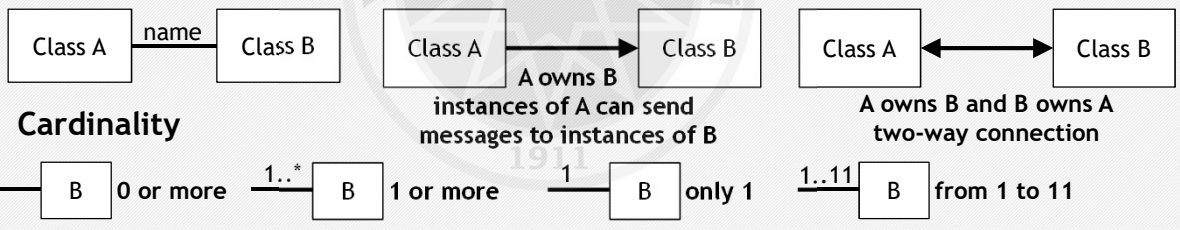
RELATIONSHIPS BETWEEN OBJECTS AND CLASSES

- Association
- Dependency
- Aggregation
- Composition
- Inheritance

ASSOCIATION

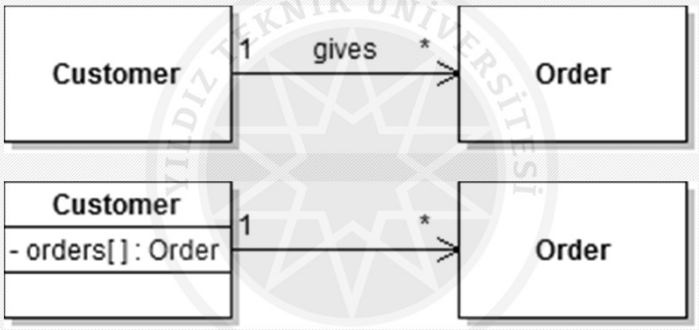
- The essence of association is ownership (SAHİPLİK).





Cardinality

ASSOCIATION - HIDDEN INFORMATION

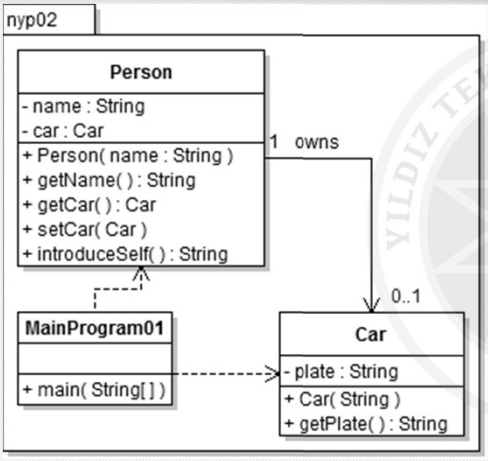


DEPENDENCY

- The essence of dependency is either **being a parameter of a method** or **temporary usage**, without ownership.

A depends on B
instances of A can send messages to instances of B in the body of aMethod.

ASSOCIATION – ONE WAY

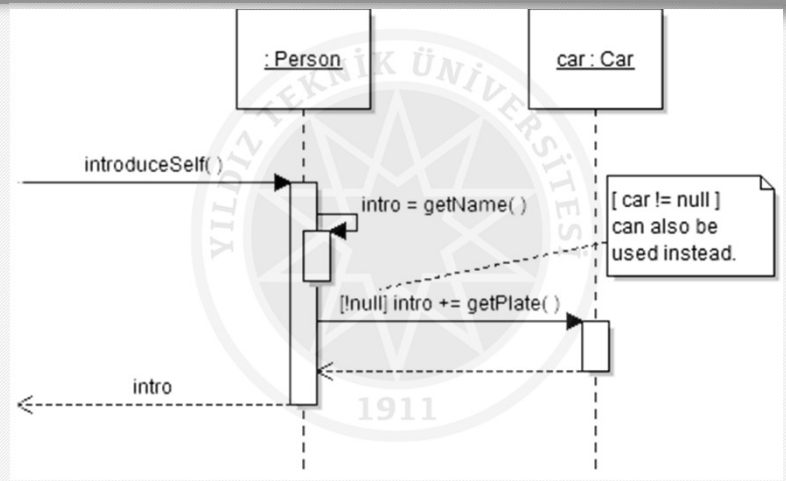



```
package nyp02;
public class Person {
    private String name;
    private Car car;

    public Person( String name ) {
        this.name = name;
    }
    public String getName( ) {
        return name;
    }
    public void setCar( Car car ) {
        this.car = car;
    }
    public Car getCar( ) {
        return car;
    }
    public String introduceSelf( ) {
        String intro;
        intro = "Hello, my name is " + getName();
        if( car != null )
            intro += "and I have a car with license plate "
            + car.getPlate()+ ".";
        return intro;
    }
}
```

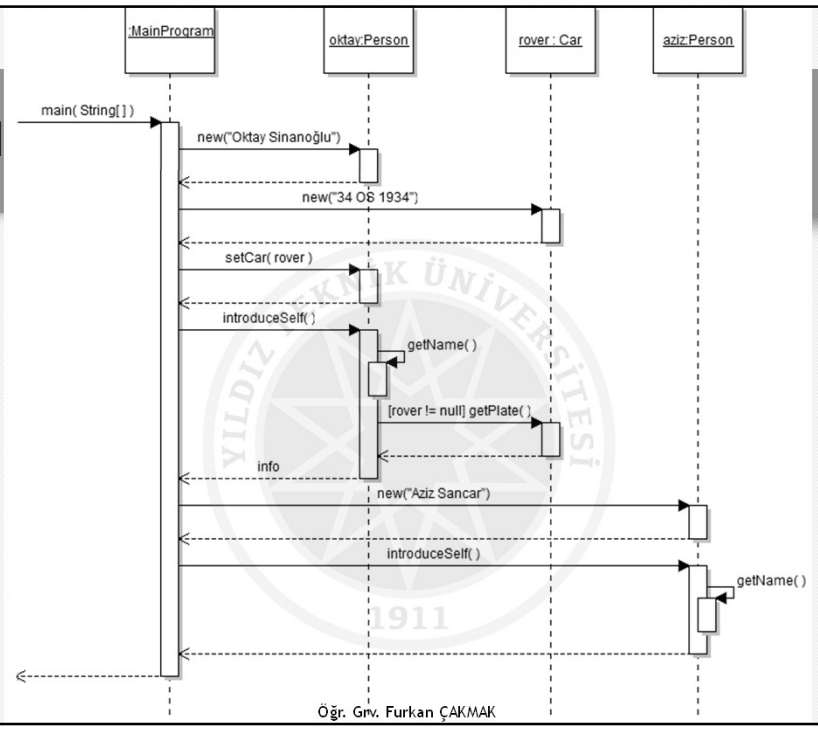

ASSOCIATION – Sequence Diagram

BLM2012
Nesneye
Yönelik
Programlama
Hafta 3

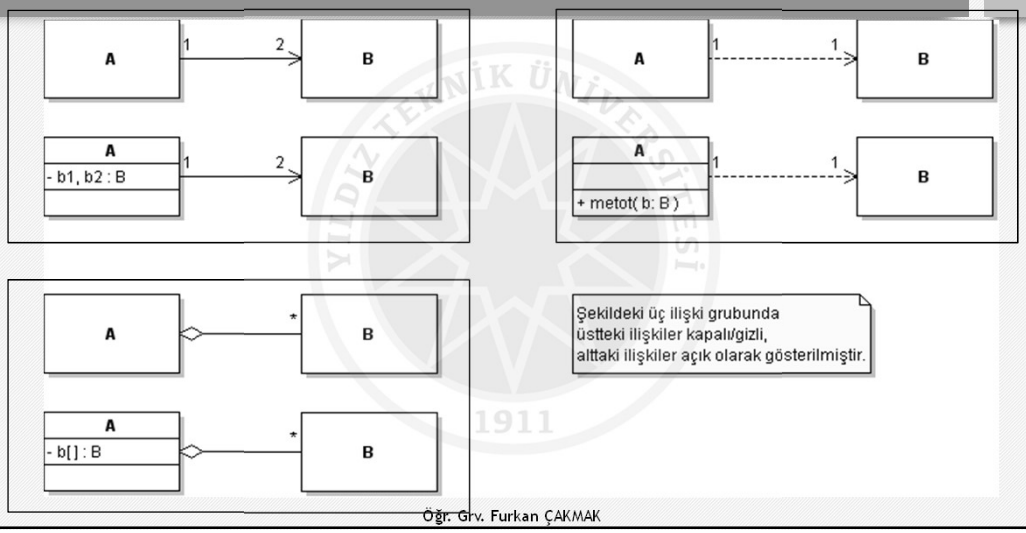


Öğr. Grv. Furkan ÇAKMAK

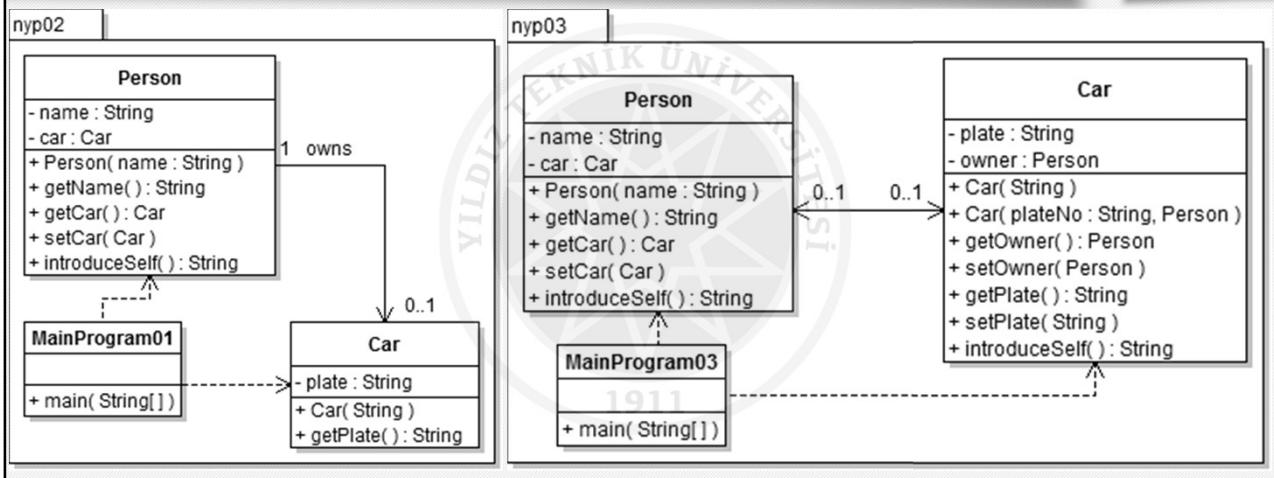
ASSOCI



ASSOCIATION – HIDDEN INFORMATION



ASSOCIATION – TWO WAY



ASSOCIATION – TWO WAY (Devam)

BLM2012
Nesneye
Yönelik
Programlama
Hafta 3

```
package nyp03;
public class Car {
   private String plate;
   private Person owner;

   public Car( String plate ) { this.plate = plate; }
   public Car( String plate, Person owner ) {
     this.plate = plate;
     this.owner = owner;
   }
   public void setOwner( Person owner ) { this.owner = owner; }
   public Person getOwner() { return owner; }
   public String getPlate( ) { return plate; }
   public void setPlate( String plate ) { this.plate = plate; }
   public String introduceSelf( ) {
     String intro;
     intro = "[CAR] My license plate is " + getPlate();
     if( owner != null )
          intro += " and my owner is " + owner.getName();
     return intro;
   }
}
```

```
package nyp03;
public class MainProgram02 {
public static void main(String[] args) {
     Person oktay = new Person("Oktay Sinanoğlu");
     Car rover = new Car("06 OS 1934");
     oktay.setCar(rover);
     rover.setOwner(oktay);
     System.out.println( oktay.introduceSelf() );
     System.out.println( rover.introduceSelf() );

     Person aziz = new Person("Aziz Sancar");
     Car honda = new Car("47 AZ 1946");
     aziz.setCar(honda);
     honda.setOwner(aziz);
     System.out.println( aziz.introduceSelf() );
     System.out.println( honda.introduceSelf() );
   }
}
```

?

Öğr. Grv. Furkan ÇAKMAK

ASSOCIATION – TWO WAY (Devam)


```
package nyp03b;
public class Person {
      /*the rest is the same*/
      public void setCar( Car car ) {
            this.car = car;
            if( car.getOwner() != this )
                  car.setOwner(this);
      }
}

package nyp03b;
public class Car {
      /*the rest is the same*/
      public void setOwner( Person owner ) {
            this.owner = owner;
            if( owner.getCar() != this )
                  owner.setCar(this);
      }
}
```

Attention!

Attention!

ASSOCIATION – TWO WAY (Devam)


```
package nyp03c;
public class Car {
   private String plate;
   private Person owner;

   public Car( String plate ) { this.plate = plate; }
   public Car( String plate, Person owner ) {
     this.plate = plate;
     setOwner(owner);
   }
   public void setOwner( Person owner ) { this.owner = owner; }
   public Person getOwner() { return owner; }
   public String getPlate( ) { return plate; }
   public void setPlate( String plate ) { this.plate = plate; }
   public String introduceSelf( ) {
     String intro;
     intro = "[CAR] My license plate is " + getPlate();
     if( owner != null )
          intro += " and my owner is " + owner.getName();
     return intro;
   }
}
```

Attention!

```
package nyp03c;
public class MainProgram03 {
     public static void main(String[] args) {
          Person oktay = new Person("Oktay Sinanoğlu");
          Car rover = new Car("06 OS 1934", oktay);
          System.out.println( oktay.introduceSelf() );
          System.out.println( rover.introduceSelf() );
     }
}
```

Sabırla Dinlediğiniz İçin Teşekkürler

BLM2012
Nesneye
Yönelik
Programlama
Hafta 3

Öğr. Grv. Furkan Çakmak

Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

Ders Tanıtım Formu ve Konular

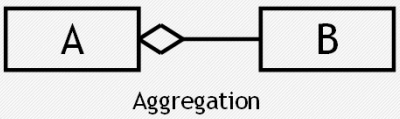
Hafta	Tarih	Konular
1	01.03.2023	Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavramları
2	08.03.2023	UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması
3	15.03.2023	Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları
4	22.03.2023	Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları
5	29.03.2023	Toplama ve Meydana Gelme İlişkileri
6	05.04.2023	Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı
7	12.04.2023	Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi)
8	19.04.2023	Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları)
9	26.04.2023	1. Ara Sınav
10	03.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar
11	10.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar (Devam)
12	17.05.2023	Paralel Programlamaya Giriş
13	24.05.2023	2. Ara Sınav
14	31.05.2023	Grafik User Interface (GUI) Giriş

RELATIONSHIPS BETWEEN OBJECTS AND CLASSES

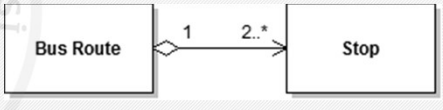
- **Association**
- **Dependency**
- Aggregation
- Composition
- Inheritance

AGGREGATION

- Represents a weak **whole-part relationship**
- **A instances has multiple B instances.**
- **A: Whole, B: Part.**
- Even if it is not shown in the diagrams, aggregation implies the following:
  - 1 on the diamond end
  - * (multiplicity) and arrow on the other end
- Agrregation is stronger than association, but only conceptually.
  - It implies that this relation has stronger rules than a regular association.
  - For example, a bus route consists of at least 2 stops and there are rules for adding a new stop to a route.

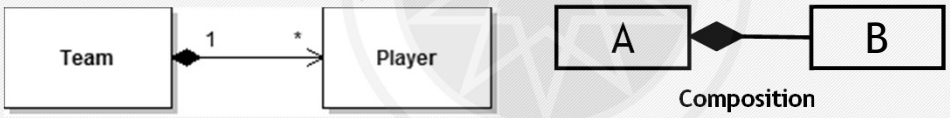


Aggregation



COMPOSITION

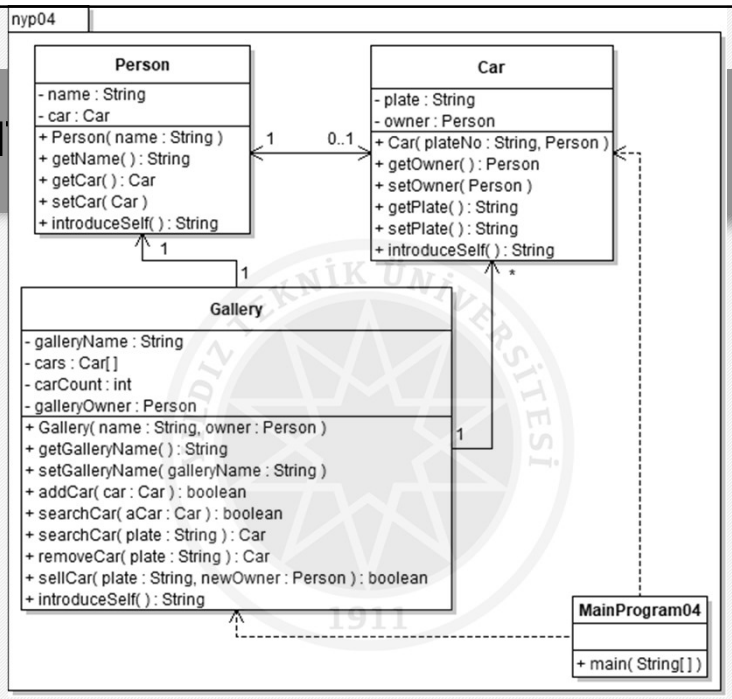
- Similar to aggregation, but represents a **stronger whole-part** relation.
- The strength of composition over aggregation is that in composition, the part can only belong to one whole at the same time.



1..*
ASSOCIATION, AGGREGATION AND COMPOSITION

- Implementation of 1..* association, aggregation and composition is similar.
- In order to implement multiplicity, we must choose a data structure:
  - Arrays, lists, stacks, queues, heaps, trees, graphs, etc.

COMPOSI



Sabırla Dinlediğiniz İçin Teşekkürler

BLM2012
Nesneye
Yönelik
Programlama
Hafta 4

Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

Ders Tanıtım Formu ve Konular

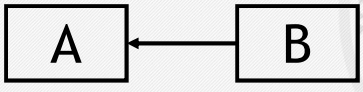
Hafta	Tarih	Konular
1	01.03.2023	Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavramları
2	08.03.2023	UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması
3	15.03.2023	Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları
4	22.03.2023	Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları
5	29.03.2023	Toplama ve Meydana Gelme İlişkileri
6	05.04.2023	Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı
7	12.04.2023	Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi)
8	19.04.2023	Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları)
9	26.04.2023	1. Ara Sınav
10	03.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar
11	10.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar (Devam)
12	17.05.2023	Paralel Programlamaya Giriş
13	24.05.2023	2. Ara Sınav
14	31.05.2023	Grafik User Interface (GUI) Giriş

RELATIONSHIPS BETWEEN OBJECTS AND CLASSES

- **Association**
- **Dependency**
- Aggregation
- Composition
- Inheritance

INHERITANCE

- Real world: A child inherits genetic properties from his/her parents.
- OOP: A means of creating new classes from an existing class, in a way that is similar with the real world.



Inheritance

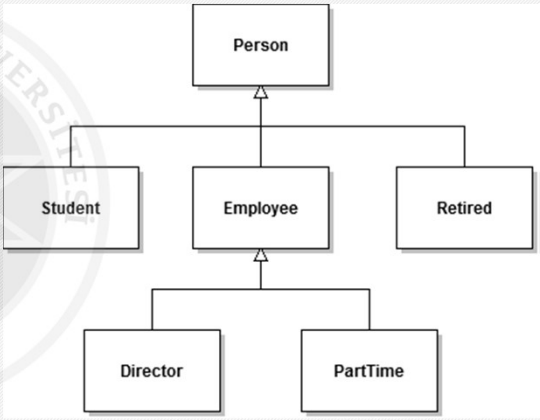
Parent class
Super class
Base class

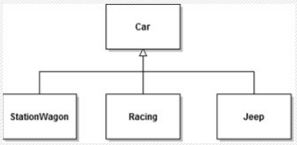
Child class
Sub class
Derived class

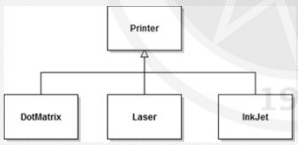
- private
 # protected

INHERITANCE (Devam)

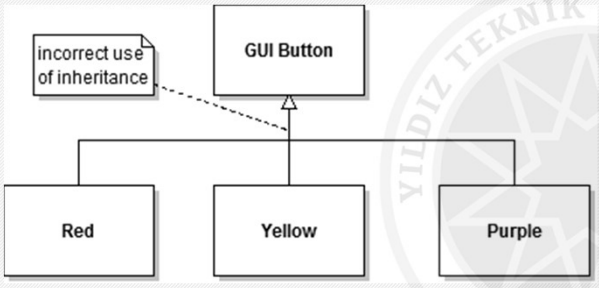
- Override
  - Final methods cannot be overridden.
- New members can be added to sub classes.
- Inheritance hierarchy / Inheritance tree
- Inheritance is also called as the generalization – specialization relation

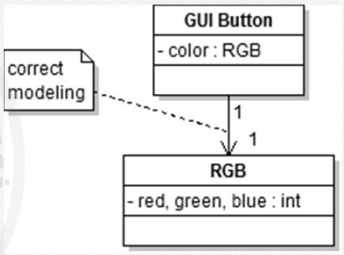




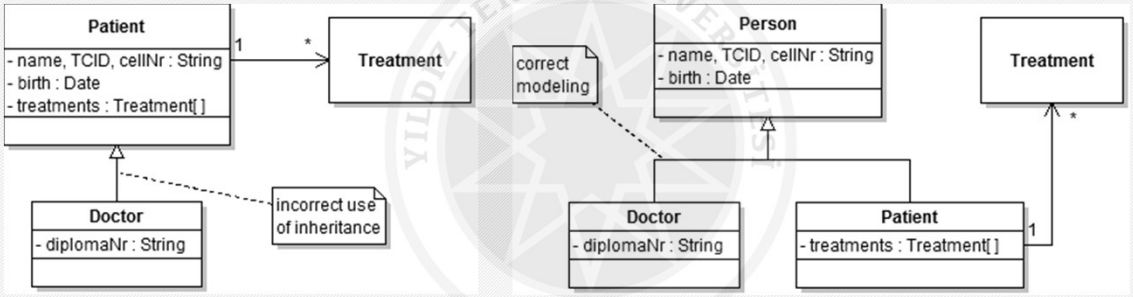


Misuse of Inheritance





Misuse of Inheritance (Devam)

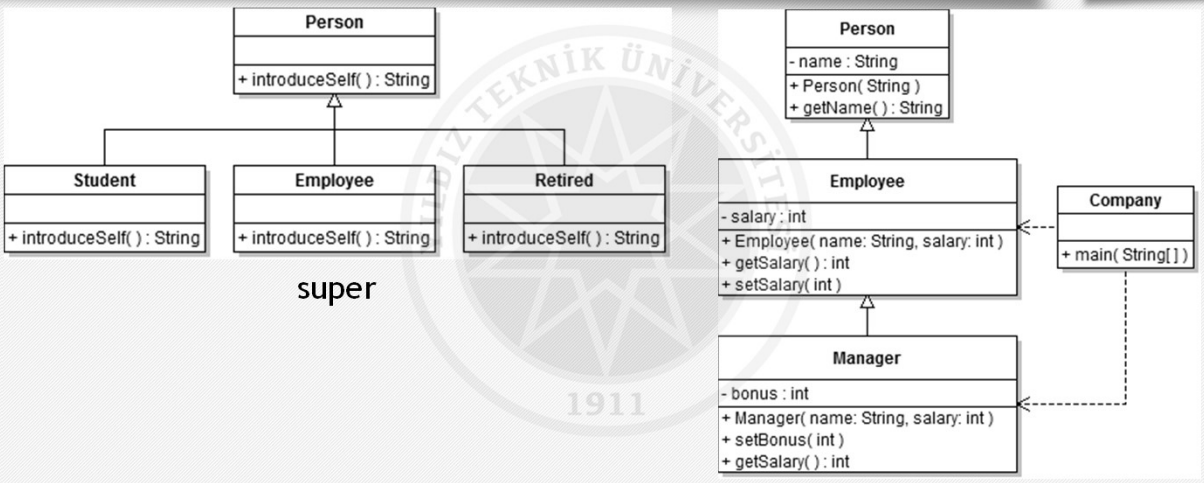


INHERITANCE (Devam)

- Evaluation of inheritance relationship:
  - Inheritance is a class-level relationship that cannot be changed during runtime.
  - Experienced object oriented modelers choose inheritance only if there is a clear gen-spec. relationship. Otherwise, association, composition or aggregation (object-level relationships) is used.
  - Correct use of inheritance is an important part of every design pattern, a subject that must be studied by people who want to be software professionals after complete study of object orientation.

POLYMORPHISM and OVERRIDING

BLM2012
Nesneye
Yönelik
Programlama
Hafta 5



Sabırla Dinlediğiniz İçin Teşekkürler

BLM2012
Nesneye
Yönelik
Programlama
Hafta 4

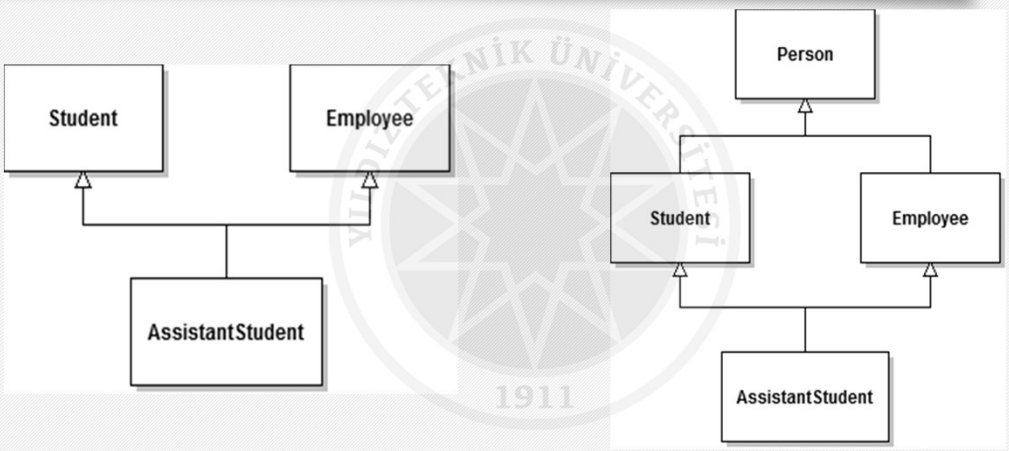
Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

Ders Tanıtım Formu ve Konular

Hafta	Tarih	Konular
1	01.03.2023	Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavramları
2	08.03.2023	UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması
3	15.03.2023	Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları
4	22.03.2023	Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları
5	29.03.2023	Toplama ve Meydana Gelme İlişkileri
6	05.04.2023	Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı
7	12.04.2023	Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi)
8	19.04.2023	Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları)
9	26.04.2023	1. Ara Sınav
10	03.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar
11	10.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar (Devam)
12	17.05.2023	Paralel Programlamaya Giriş
13	24.05.2023	2. Ara Sınav
14	31.05.2023	Grafik User Interface (GUI) Giriş

Multiple Inheritance



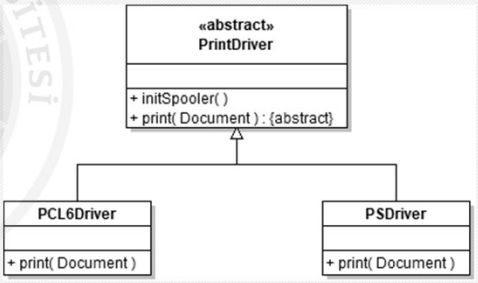
Abstract Classes

- An abstract class is such a class that it is used as a base class and it represents a template for its regular sub classes.
  - If a class is abstract, we identify it with the keyword abstract.
- It is forbidden to create instances of an abstract class.
- Abstract classes can have member fields, just like the concrete classes.
- Abstract classes can have both concrete and abstract member methods.
  - An abstract method has only definition together with the keyword abstract, it does not have a body.

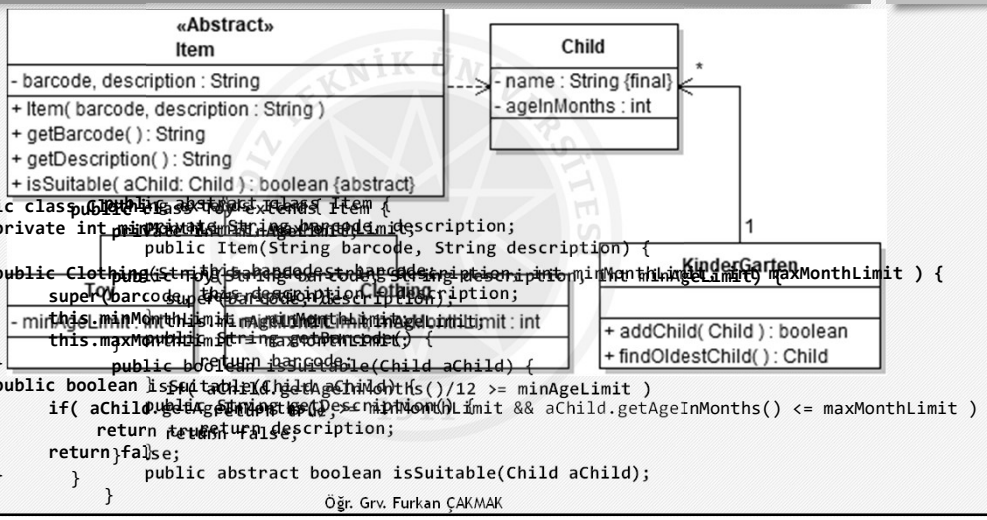
Abstract Classes (Con't)

- When do we need abstract classes?
- You can mark the abstract classes in UML class schemas in italics or by adding the <<Abstract>> stereotype.
  - <<…>>: This is called a stereotype.

```
public abstract class PrintDriver {
    public void initSpooler( ) {
    /* necessary codes*/
     }
     public abstract void print( Document doc );
}
public class PCL6Driver extends PrintDriver {
    public void print(Document doc) {
    //necessary code is inserted here
     }
}
```

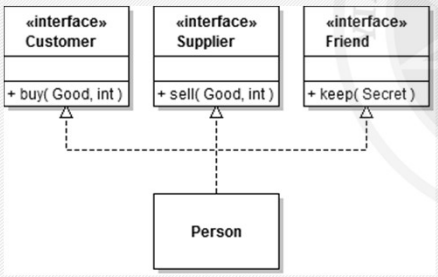


Abstract Classes: Example



Interfaces

- Interfaces can be thought as abstract classes **without members.**
  - **If you wish, you may add "public final static" member fields only.**
- An interface is a named collection of methods.
- UML representation and source code of an example:



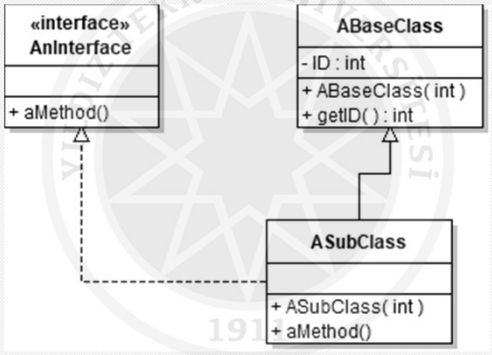

```
public interface Customer {
    public void buy( Good aGood, int quantity );
}
public interface Supplier {
    public void sell( Good aGood, int quantity );
}
public interface Friend {
    public void keep( Secret aSecret );
}
public class Person implements Customer,
        Supplier, Friend {
    public void buy( Good aGood, int quantity ) {
        //related code
    }
    public void sell (Good aGood, int quantity ) {
        // related code
    }
   public void keep( Secret aSecret ) {
        // related code
    }
}
```


Interfaces (Con't)

- We use interfaces …
  - in order to group responsibilities of entities,
  - in order to give objects multiple views,
  - instead of inheritance,
    - Because inheritance is a "heavy weight" relation that should be used only when it is absolutely necessary.
  - instead of multiple inheritance.
- Rules related to interfaces:
  - A class should code the bodies of all the methods of the implemented interfaces.
  - Regular member fields cannot be defined in interfaces. Interfaces can only have "public final static" member fields.
  - Only public methods can be defined in interfaces.
  - Interfaces cannot have constructors.
  - A class can implement multiple interfaces.
  - Suggestion: Begin naming interfaces with I (capital i).

Interfaces (Con't)

- Interface implementation and inheritance can be used together when needed:



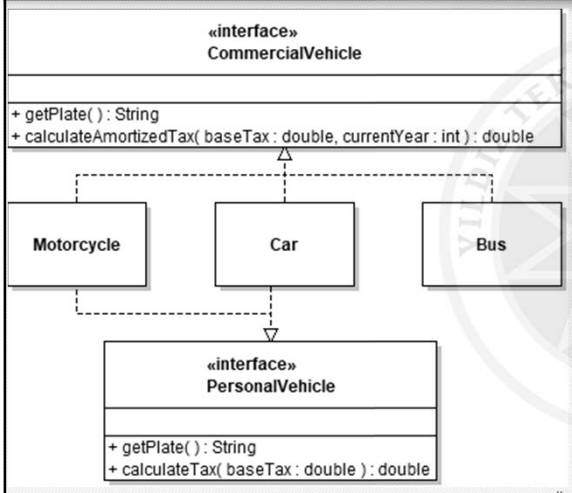

```
public class ASubClass extends ABaseClass implements AnInterface {
    // it should be easy to code the rest for you
}
```


DESIGNING AND CODING INTERFACES

- Consider the following requirement about calculating the taxes of vehicles:
  - Taxation of commercial and personal vehicles is different.
  - Motorcycles, cars and buses can be registered as commercial vehicles.
  - Only motorcycles and cars can be registered as personal vehicles .
  - Only taxes of commercial vehicles can be amortized.
  - Commercial or not, calculation of the tax of different vehicles (car, bus, etc.) are very different.
- How can we model this requirement?

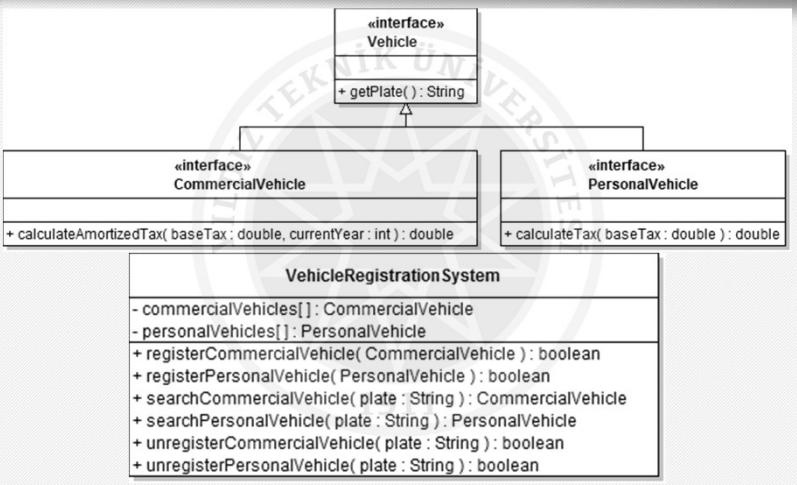
- Hint: If the tax calculation for different vehicles were similar (i.e. parametrized), using one abstract base class instead of interfaces would be a better choice.
- PS: We will code a year limit for maximum amortizement in the CommercialVechile interface but VioletUML doesn't let us draw this in here.

DESIGNING AND CODING INTERFACES (CON'T)




```
public interface CommercialVehicle {
    public final static int yearLimit = 10;
    public String getPlate();
    public double calculateAmortizedTax(double
        baseTax, int currentYear );
}
public interface PersonalVehicle {
    public String getPlate();
    public double calculateTax( double baseTax);
}
```


DESIGNING AND CODING INTERFACES (CON'T)



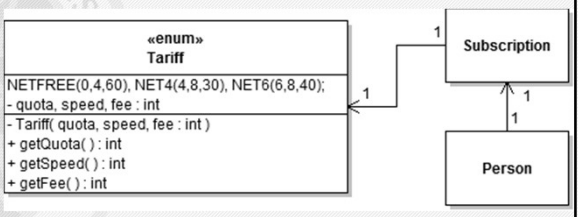
PRIMITIVE ENUMERATIONS and ENUM CLASSES


```
public enum Size {
    SMALL, MEDIUM, LARGE, EXTRA_LARGE;
}

Size s = Size.MEDIUM;
```

```
    public enum Tariff {
        NETFREE(0,4,60), NET4(4,8,30), NET6(6,8,40);
        private int quota, speed, fee;
        private Tariff( int quota, int speed, int fee ) {
            this.quota = quota; this.speed = speed; this.fee = fee;
        }
        public int getQuota() { return quota; }
        public int getSpeed() { return speed; }
        public int getFee() { return fee; }
    }

    public class Test {
        public static void main(String[] args) {
            Tariff tariff4 = Tariff.NET4;
            Person yunus = new Person("Yunus Emre");
            yunus.subscribeTo(tariff4);
            Person berkin = new Person("Berkin Gülay");
            berkin.subscribeTo(Tariff.NETFREE);
            System.out.println(yunus);
            System.out.println(berkin);
        }
    }
```



Sabırla Dinlediğiniz İçin Teşekkürler

BLM2012
Nesneye
Yönelik
Programlama
Hafta 6

Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

Ders Tanıtım Formu ve Konular

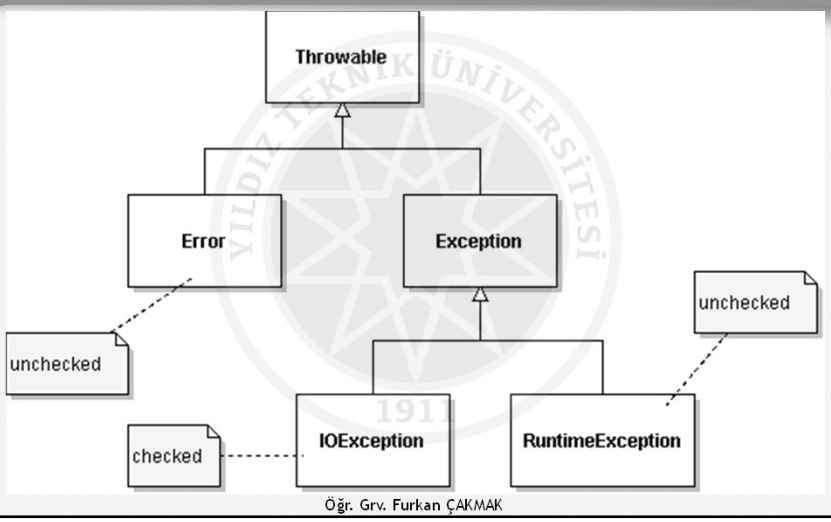
Hafta	Tarih	Konular
1	01.03.2023	Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavramları
2	08.03.2023	UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması
3	15.03.2023	Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları
4	22.03.2023	Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları
5	29.03.2023	Toplama ve Meydana Gelme İlişkileri
6	05.04.2023	Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı
7	12.04.2023	Hata Ayıklama (Exception Handling), Birim Testler (Unit Test)
8	19.04.2023	Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi)
9	26.04.2023	1. Ara Sınav
10	03.05.2023	Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları)
11	10.05.2023	Tip dönüşümü, Enum Sınıfları, İç Sınıflar
12	17.05.2023	Paralel Programlamaya Giriş
13	24.05.2023	2. Ara Sınav
14	31.05.2023	Grafik User Interface (GUI) Giriş

EXCEPTION HANDLING

- Some sources of error are:
  - Bugs in JVM
  - Wrong input by the user
  - Buggy code written by us
  - Acts of God
    - A lone and humble programmer cannot control:
      - every aspect of Internet traffic,
      - file access rights,
      - etc.
    - But we should be aware of them and deal with them!
- There are multiple ways of dealing with errors.
  - Boolean returns
  - Form components with error checking mechanisms
  - Exception handling.
- Exception handling is a form of error trapping.

EXCEPTION HANDLING

BLM2012
Nesneye
Yönelik
Programlama
Hafta 7



EXCEPTION HANDLING

- java.lang.Error:
  - indicates serious problems that a reasonable application should not try to catch
    - Internal JVM bugs, etc.
    - java.lang.UnsupportedClassVersionError: Can happen when you move your code between different versions of Eclipse/IDE.
- java.lang.RuntimeException:
  - This is mostly caused by our buggy code
    - java.lang.NullPointerException: We have tried to use an uninitialized object
    - java.lang.IndexOutOfBoundsException: We have tried to access a non-existent member of an array.
    - etc.
- java.io.IOException:
  - Something went wrong during a file operation or a network operation.
  - These operations are always risky, so we must have an alternate plan in case of something goes wrong.
    - If having an alternate plan is a must, than the exception is determined as checked.

EXCEPTION HANDLING

BLM2012
Nesneye
Yönelik
Programlama
Hafta 7

- Handling checked exceptions is done by coding a try - catch block.

```
try {
    /* error-prone methods */;
}
catch( AnException e ) {
    /* Dealing with error */
}
```

- A programmer may opt to not handle a checked exception.
  - However, someone will eventually handle it!

```
aMethod(…) throws AnException {
    /* error-prone methods */
}
```

EXCEPTION HANDLING

- It is possible to handle multiple exceptions as well:

```
try {
    /* error-prone methods */;
}
catch( AnException e ) {
    /* Dealing with error */
}
catch( AnotherException e ) {
    /* Dealing with error */
}
```


EXCEPTION HANDLING

- What should I do in a catch block?
  - Inform the user about the error with the e.printStackTrace( ) method.
  - Log this error
- If this is a very serious error, you may release some resources and make a "clean exit" in the finally block.
  - Scopes of the try block and the finally block are different. Therefore you cannot access the temporary variables/objects defined in the try block from the finally block. Plan your "clean exit" accordingly.
  - The finally block executes whether an exception is thrown or not.

```
try {
    /* error-prone methods */;
}
catch( AnException e ) {
    /* Dealing with error */
}
catch( AnotherException e ) {
    /* Dealing with error */
}
finally {
    /* make a clean exit */
}
```

EXCEPTION HANDLING


```
public class ExceptionExample01 {
    MyScreenRenderer graphics;
    MyCADfile myFile;
    //Other methods of this class are omitted
    public void parseMyCADfile( String fileName ) {
        try {
         graphics = new MyScreenRenderer();
         myFile = openFile( fileName );
         MyFigure figs[ ] = myFile.readFromFile( );
         drawFigures( figs );
         myFile.close();
        }
        catch( IOException e ) {
         System.out.println("An IO exception has occurred"+
          " while opening or reading from file "+fileName+":"
          + e.toString( ) );
         e.printStackTrace( );
         System.exit(1); //Multithreaded, allows finally to be run
        }
        finally {
         graphics.releaseSources();
        }
    }
}
```


EXCEPTION HANDLING

BLM2012
Nesneye
Yönelik
Programlama
Hafta 7

- You can create your own Exception classes by :
  - inheriting from IOException if you want your exception to be a checked one,
  - inheriting from RuntimeException if you want an unchecked one.

```
public class MyFileFormatException extends IOException {
    public MyFileFormatException( ) {
        super( );
    } //was required in JDK versions older than 5
    public MyFileFormatException( String errorMessage ) {
        super( errorMessage );
        /* Other things to do (optional) */
    } //necessary for informing the user and/or programmer
}
```

EXCEPTION HANDLING

- Throwing an exception:
  - If something terrible may happen during your code, you can throw an exception

```
public class AProgram {
    public void processFile ( ) throws MyFileFormatException{
        some_statements();
        if( an_unexpected_situation )
            throw new MyFileFormatException("... happened");
    }
}
```


EXCEPTION HANDLING - EXAMPLE


```
package nyp09;
import java.io.IOException;
/* @home: Check the needed additions
 * if we had extended this exception
 * from java.lang.RuntimeException
 */
@SuppressWarnings("serial")
public class ImpossibleInfo extends IOException {
       public ImpossibleInfo( String errorMessage ) {
           super(errorMessage);
       }
   }
}
```

EXCEPTION HANDLING – EXAMPLE CON'T


```
package nyp09;
public class Person {
    private String name;
    private int age;

    public Person( String name ) { this.name = name; }
    public String getName( ) { return name; }
    public int getAge( ) { return age; }
    public String toString() {
        return getName() + " " + getAge( );
    }
    public void setAge( int age ) throws ImpossibleInfo {
        if( age < 0 || age > 150 )
           throw new ImpossibleInfo("Impossible age: "+age);
        this.age = age;
    }
}
```


EXCEPTION HANDLING – EXAMPLE CON'T

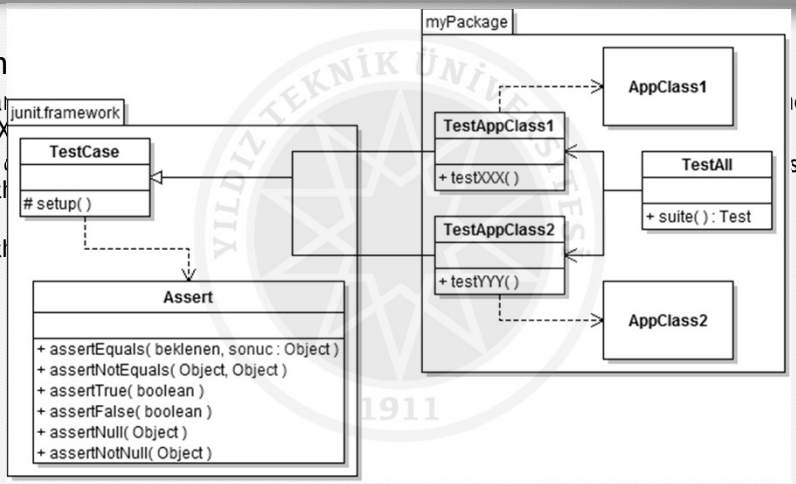

```
package nyp09;
import java.util.*;
public class TestExceptions {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.print("Enter person's name: ");
        String name = in.nextLine();
        Person insan = new Person(name);
        try {
            System.out.print("Enter age: ");
            int age = in.nextInt();
            insan.setAge(age);
            System.out.println(insan);
        }
        catch (ImpossibleInfo e) {
            e.printStackTrace();
        }
        finally {
            in.close();
        }
    }
}
```

UNIT TESTING – ABOUT UNIT TESTING WITH TOOL SUPPORT

- We have used a separate class having a main method to test our code so far.
  - We have tested the responsibilities of our smallest coding unit, namely the classes we have coded.
  - This is called **'Unit Testing'** in literature.
- Notice that we had to use extensive if-else cases to determine whether a test case was successful or not.
  - Then we had to analyze all the printouts why and where a test case has failed.
- We need to be able to design, execute and analyze the results of our tests.
  - Having a tool for this purpose helps a lot.
- A widely used unit testing tool named **jUnit** can help us.

UNIT TESTING WITH JUNIT

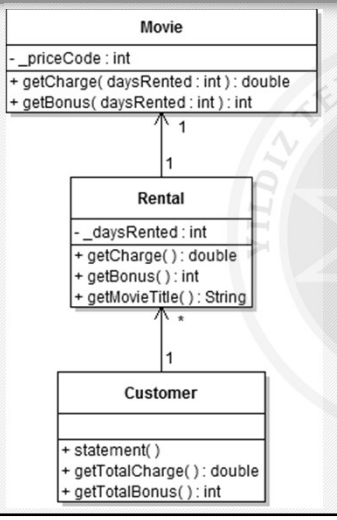
- Preparin
  - Prepa ethods such as testXX
    - You stAll.suite( )
    - The

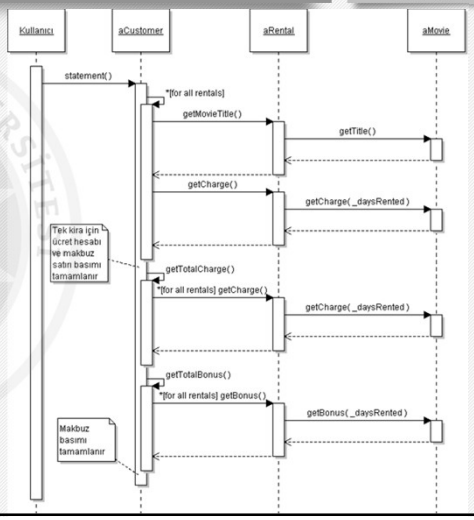


UNIT TESTING

- Preparing test cases with jUnit version 4.X:
  - In addition to preserving backwards compatibility with v3, jUnit v4 adds annotation support:
    - Test classes are no longer needed to inherit from the TestCase class.
    - Test case methods' names no longer need to start with the test word, it is enough to annotate them by using the @test annotation.
      - The setup method is annotated by @before.
      - `@Before`
      - `public void setUp() { /*Preparations*/ }`
      - `@Test`
      - `public void testSomething() { /*Do test*/ }`
    - Exception support is now possible, too: You can test whether a necessary exception is thrown or not, without halting the tests.
      - `@Test(expected=SomeException.class)`
      - `public void testTheException() throws Exception {`
      - `doSomethingThatCreatesTheException();`
      - `}`

UNIT TESTING -
Let's code unit test cases for a simple movie rental example





UNIT TESTING -
Let's code unit test cases for a simple movie rental example


```
package unitTestingByFowler;
import junit.framework.TestCase;
public class TestCustomer extends TestCase {
   private Customer yunus;
   private Movie matrix, monster, surrogate, terminator;
   protected void setUp() {
     yunus = new Customer("Yunus Emre Selçuk");
     matrix = new Movie("The Matrix",Movie.REGULAR);
     monster = new Movie("Monsters, Inc.",Movie.CHILDRENS);
     surrogate = new Movie("Surrogates", Movie.NEW_RELEASE);
     terminator = new Movie("Terminator Salvation",Movie.NEW_RELEASE);
   }
   public void testGetName( ) {
     String sonuc = yunus.getName( );
     assertEquals("Yunus Emre Selçuk",sonuc);
   }
   public void testStatementWhenEmpty( ) {
     String sonuc = yunus.statement();
     String beklenen = "Rental Record for Yunus Emre Selçuk\n";
     beklenen += "Amount owed is 0.0\n";
     beklenen += "You earned 0 frequent renter points";
     assertEquals(beklenen, sonuc);
   }
```

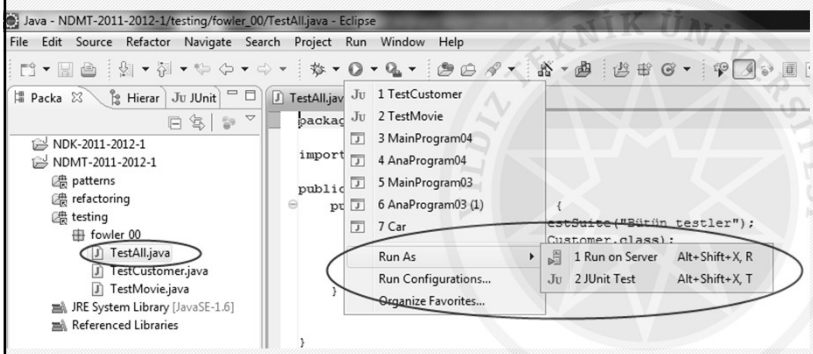

UNIT TESTING -
Let's code unit test cases for a simple movie rental example

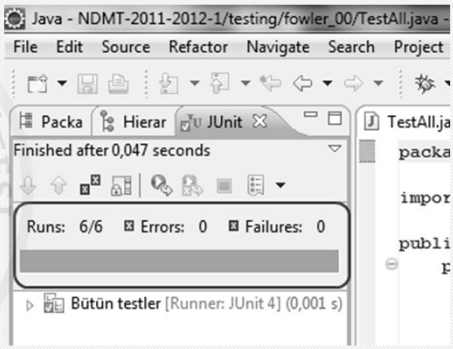

```
public void testStatementWithMoviesLongRent( ) {
  yunus.addRental( new Rental(matrix, 3) );
  yunus.addRental( new Rental(monster, 4) );
  yunus.addRental( new Rental(surrogate, 2) );
  String sonuc = yunus.statement();
  String beklenen = "Rental Record for Yunus Emre Selçuk\n";
  beklenen += "\tThe Matrix\t3.5\n";
  beklenen += "\tMonsters, Inc.\t3.0\n";
  beklenen += "\tSurrogates\t6.0\n";
  beklenen += "Amount owed is 12.5\n";
  beklenen += "You earned 4 frequent renter points";
  assertEquals(beklenen, sonuc);
}
public void testStatementWithMoviesShortRent( ) {
  yunus.addRental( new Rental(matrix, 2) );
  yunus.addRental( new Rental(monster, 3) );
  yunus.addRental( new Rental(surrogate, 1) );
  String sonuc = yunus.statement();
  String beklenen = "Rental Record for Yunus Emre Selçuk\n";
  beklenen += "\tThe Matrix\t2.0\n";
  beklenen += "\tMonsters, Inc.\t1.5\n";
  beklenen += "\tSurrogates\t3.0\n";
  beklenen += "Amount owed is 6.5\n";
  beklenen += "You earned 3 frequent renter points";
  assertEquals(beklenen, sonuc);
}
```

UNIT TESTING -
Let's code unit test cases for a simple movie rental example


```
 public void testNewReleaseRentalBonus( ) {
     yunus.addRental( new Rental(surrogate, 2) );
     yunus.addRental( new Rental(terminator, 1) );
     String sonuc = yunus.statement();
     String beklenen = "Rental Record for Yunus Emre Selçuk\n";
     beklenen += "\tSurrogates\t6.0\n";
     beklenen += "\tTerminator Salvation\t3.0\n";
     beklenen += "Amount owed is 9.0\n";
     beklenen += "You earned 3 frequent renter points";
     assertEquals(beklenen, sonuc);
    }
}
package unitTestingByFowler;
import junit.framework.*;
public class TestAll {
   public static Test suite( ) {
      TestSuite suite = new TestSuite("Bütün testler");
      suite.addTestSuite(TestCustomer.class);
      suite.addTestSuite(TestMovie.class);
      return suite;
   }
}
```


UNIT TESTING -
Let's code unit test cases for a simple movie rental example





UNIT TEST with JUNIT

- Disadvantages of manual testing:
  - Our test code is one overly long function that is harder to examine.
    - We may forget what we are testing about.
  - We may need to manually search any "Problem" string in a long and verbose output text.
  - Our problem cases so far did not tell what especially gone wrong.
- We can alleviate these problem by automated test execution and evaluation.
  - Our test code becomes more modular
  - We have a green bar instead! Moreover, assertEquals compares its two parameters and highlight the first difference between them, especially they are String instances.

Sabırla Dinlediğiniz İçin Teşekkürler