# Nesneye Yönelik Programlama BLM2012

Öğr. Grv. Furkan ÇAKMAK

---

## Ders Tanıtım Formu ve Konular
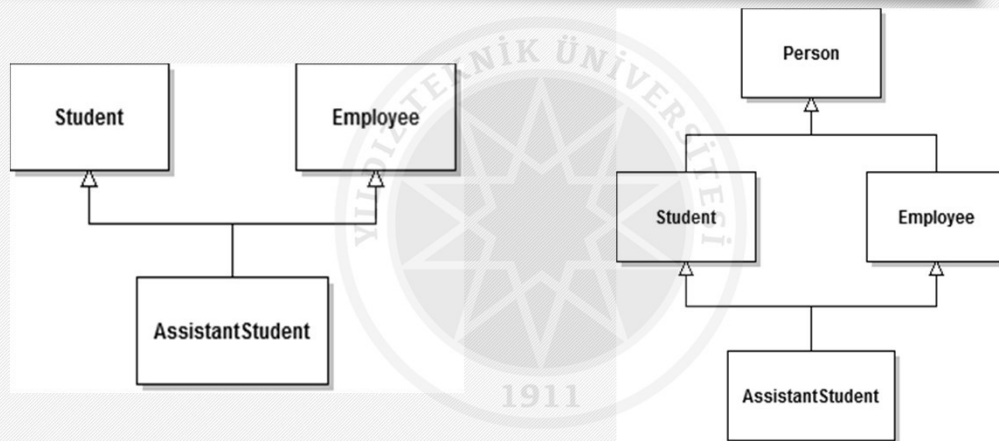
BLM2012
Nesneye
Yönelik
Programlama
Hafta 6

| Hafta | Tarih | Konular |
|---|---|---|
| 1 | 01.03.2022 | Dersin ve Java Dilinin Genel Tanıtımı, Sınıflar, Nesneler, Üyeler, Final ve Static Kavraları |
| 2 | 08.03.2022 | UML Sınıf Şemaları, Kurucular ve Sonlandırıcılar, Denetim Akışı, Nesneleri Oluşturulması |
| 3 | 15.03.2022 | Kurucuların ve Metotların Çoklu Tanımlanması, İlkeller, String ve Math Sınıfları |
| 4 | 22.03.2022 | Sahiplik ve Kullanma İlişkileri, Tek Yönlü ve İki Yönlü Sahiplik Kavramları |
| 5 | 29.03.2022 | Kalıtım, Metotların Yeniden Tanımlanması ve Çoklu Metot Tanımlamadan Farkı |
| 6 | 05.04.2022 | NYP'da Özel Konular: Abstract Classes, Interfaces, Enum Sınıfları |
| 7 | 12.04.2022 | Exception Handling, Unit Test |
| 8 | 19.04.2022 | 1. Ara Sınav |
| 9 | 26.04.2022 | Temel Veri Yapılarının Jenerik Sınıflar Eşliğinde Kullanımı (Liste ve Eşleme Yapıları). |
| 10 | 03.05.2022 | Ramazan Bayramı |
| 11 | 10.05.2022 | Dosyalar ve Akışlar ile Çalışmak (Serileştirme ve Ters İşlemi) |
| 12 | 17.05.2022 | Tip dönüşüm, Enum Sınıfları, İç Sınıflar |
| 13 | 24.05.2022 | 2. Ara Sınav |
| 14 | 31.05.2022 | Paralel Progrmlamaya Giriş |

Öğr. Grv. Furkan ÇAKMAK

# Multiple Inheritance

Öğr. Grv. Furkan ÇAKMAK

---

# Abstract Classes

- An abstract class is such a class that it is used as a basee 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.
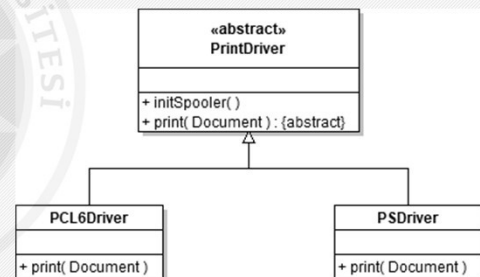
Öğr. Grv. Furkan ÇAKMAK

# 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.
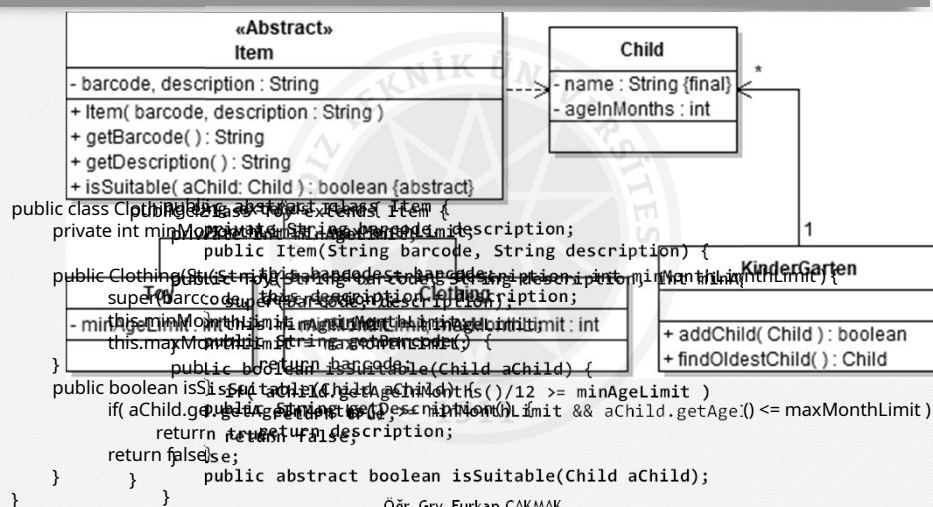  - <<...>>: This is called a stereotype.

```
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»
PrintDriver

+ initSpooler( )
+ print( Document ) : {abstract}

PCL6Driver

+ print( Document )

PSDriver

+ print( Document )

Öğr. Grv. Furkan ÇAKMAK

---

# Abstract Classes: Example

«Abstract»
Item

- barcode, description : String

+ Item( barcode, description : String )
+ getBarcode( ) : String
+ getDescription( ) : String
+ isSuitable( aChild: Child ) : boolean {abstract}

Child

- name : String {final}
- ageInMonths : int

KinderGarten

+ addChild( Child ) : boolean
+ findOldestChild( ) : Child

```
public class Clothing extends abstract class Item {
    private int minMonthLimit;
    private String barcode, description;
    public Item(String barcode, String description) {
    public Clothing(String barcode, String description, int minMonthLimit) {
        super(barcode, description);
        this.barcode = barcode;
        this.description = description;
    - minAgeLimit : int
        this.minMonthLimit = minMonthLimit;       - maxMonthLimit : int
    }                    public String getBarcode() {
                            return barcode;
    public boolean isSuitable(Child aChild) {
    }        if( aChild.getAgeInMonths()/12 >= minAgeLimit )
            public String getDescription() {
                return description;
        returrn true;
    }            }
    return false;
    }        }
                public abstract boolean isSuitable(Child aChild);
    }        }
}
```

Öğr. Grv. Furkan ÇAKMAK

# Interfaces

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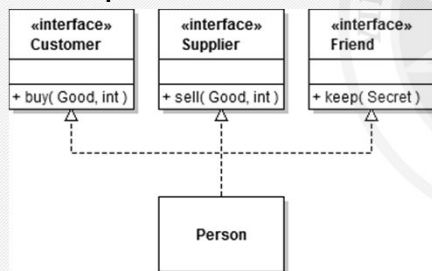


```
public interface Customer {
    public void buy( Good aGood, int quantity );
}
public interface Supplier {
    public void sell( Good aGood, int quantity );
}
public interface Friend {
    public void keep( Secret aSecret );
}
public class Person implements Customer,
        Supplier, Friend {
 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

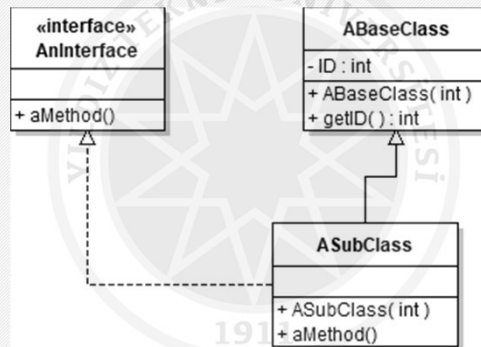Öğr. Grv. Furkan ÇAKMAK

---

# Interfaces (Con't)

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

Öğr. Grv. Furkan ÇAKMAK

# Interfaces (Con't)

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



```
public class ASubClClass extends ABaseClass implements AnInterfac
    // it should be be easy to code the rest for you
}
```

---

# DESIGNING AND CODING INTERFACES

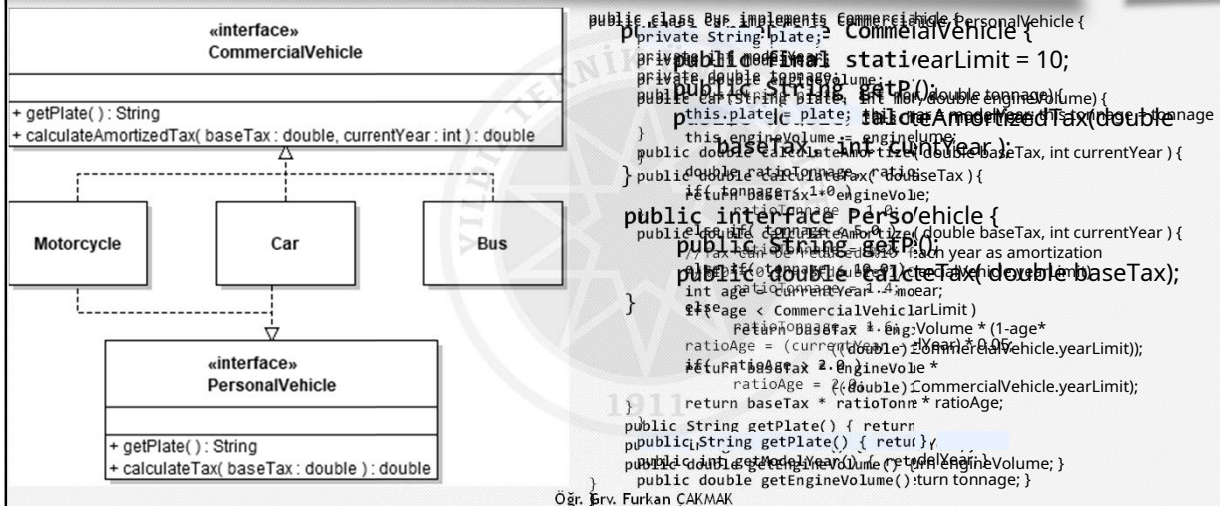- Consider the following requirement about calculating the taxes of vehicles:
  - Taxation of commercial and personal vehicles is different.
  - Motorcycles, cars and buses can be registered as commercial vehicles.
  - Only motorcycles and cars can be registered as personal vehicles .
  - Only taxes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

# DESIGNING AND CODING INTERFACES (CON'T)
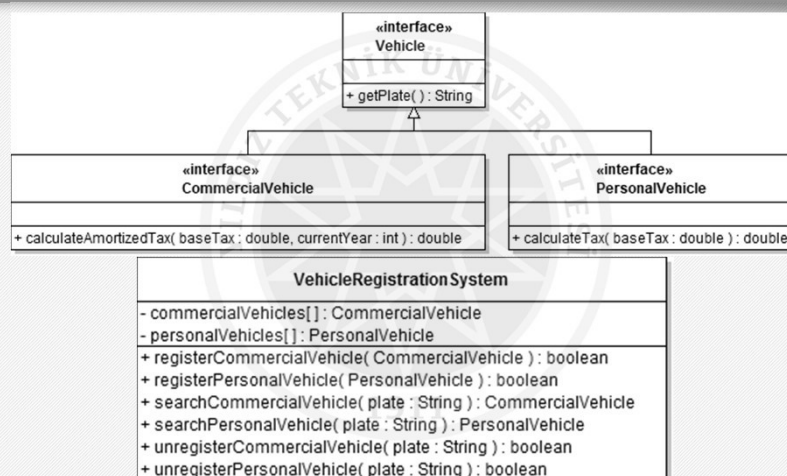
«interface»
CommercialVehicle

+ getPlate( ) : String
+ calculateAmortizedTax( baseTax : double, currentYear : int ) : double

Motorcycle     Car     Bus

«interface»
PersonalVehicle

+ getPlate( ) : String
+ calculateTax( baseTax : double ) : double

Öğr. Grv. Furkan ÇAKMAK

---

# DESIGNING AND CODING INTERFACES (CON'T)

«interface»
Vehicle

+ getPlate( ) : String

«interface»
CommercialVehicle

+ calculateAmortizedTax( baseTax : double, currentYear : int ) : double

«interface»
PersonalVehicle

+ calculateTax( baseTax : double ) : double

**VehicleRegistrationSystem**

- commercialVehicles[ ] : CommercialVehicle
- personalVehicles[ ] : PersonalVehicle
+ registerCommercialVehicle( CommercialVehicle ) : boolean
+ registerPersonalVehicle( PersonalVehicle ) : boolean
+ searchCommercialVehicle( plate : String ) : CommercialVehicle
+ searchPersonalVehicle( plate : String ) : PersonalVehicle
+ unregisterCommercialVehicle( plate : String ) : boolean
+ unregisterPersonalVehicle( plate : String ) : boolean

Öğr. Grv. Furkan ÇAKMAK

## 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);
    }
}
```

| «enum» Tariff |
|---|
| NETFREE(0,4,60), NET4(4,8,30), NET6(6,8,40); |
| - quota, speed, fee : int |
| - Tariff( quota, speed, fee : int ) |
| + getQuota( ) : int |
| + getSpeed( ) : int |
| + getFee( ) : int |

Subscription

Person
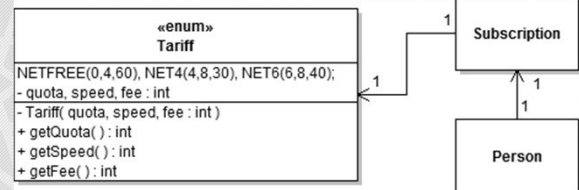
Öğr. Grv. Furkan ÇAKMAK

---

# Sabırla Dinlediğiniz İçin Teşekkürler

Teşekkürler

Öğr. Grv. Furkan Çakmak