5. Ders: JAVA ile Nesne Yönelimli Programlama Kalıtım (Inheritance)

Fırat Üniversitesi Teknoloji Fakültesi Yazılım Mühendisliği Bölümü

YMH112 Algoritma ve Programlama-II

Dr. Öğr. Üyesi Yaman Akbulut

JAVA ile Nesne Yönelimli Programlama

- http://www.kriptarium.com/algoritma.html (Yardımcı kaynak)

- JAVA ile Nesne Yönelimli Programlama (Ders: 20-30) video.

    - Ders 20: Static Anahtar Sözcüğü (izle)
    - Ders 21: Statik Değişkenler ile Program Sayacı (izle)
    - Ders 22: Statik Yöntemler (izle)
    - Ders 23: Statik Yöntemler Kullanılırken Dikkat Edilecek Unsurlar (izle)
    - Ders 24: this Anahtar Sözcüğünün Kullanımı I (izle)
    - Ders 25: this Anahtar Sözcüğünün Kullanımı II (izle)
    - Ders 26: Kalıtım/Miras (Inheritance) (izle)
    - Ders 27: Kalıtım Türleri (izle)
    - Ders 28: Java'da Neden Çoklu Kalıtım Desteklenmiyor? (izle)
    - Ders 29: Java Polimorfizmi / Yöntem Aşırı Yüklemesi (Method Overloading) (izle)
    - Ders 30: Java Polimorfizmi /Yöntem Geçersiz Kılma (Method Overriding) (izle)

Java Keywords

abstract	double	int	super
assert	else	interface	switch
boolean	enum	long	synchronized
break	extends	native	this
byte	final	new	throw
case	finally	package	throws
catch	float	private	transient
char	for	protected	try
class	goto	public	void
const	if	return	volatile
continue	implements	short	while
default	import	static	
do	instanceof	strictfp	

Kaynak: Introduction to JAVA Programming 10th Edition, Y. Daniel Liang

Kalıtım (Inheritance)

Nesne yönelimli programlama, mevcut sınıflardan yeni sınıflar tanımlamamıza olanak tanır. Buna kalıtım ya da miras denir.

Önceki derslerde anlattığımız üzere, prosedürel paradigma metotların tasarlanmasına odaklanır ve nesne yönelimli paradigma, verileri ve metotları birlikte nesneler halinde birleştirir.

Nesne yönelimli paradigmayı kullanan yazılım tasarımı, nesnelere ve nesneler üzerindeki işlemlere odaklanır.

Nesne yönelimli yaklaşım, prosedürel paradigmanın gücünü, işlemlerle verileri nesnelere entegre eden ek bir boyutla birleştirir.

Kalıtım (Inheritance)

Kalıtım, yazılımı yeniden kullanmak için önemli ve güçlü bir özelliktir.

Daireleri, dikdörtgenleri ve üçgenleri modellemek için sınıflar tanımlamamız gerektiğini varsayalım.

Bu sınıfların birçok ortak özelliği vardır.

Fazlalıktan kaçınmak ve sistemin anlaşılmasını ve bakımını kolaylaştırmak için bu sınıfları tasarlamanın en iyi yolu nedir?

Cevap, kalıtımı kullanmaktır.

Üst sınıflar ve alt sınıflar (Superclasses and subclasses)

Kalıtım, genel bir sınıf (yani bir üst sınıf) tanımlamamıza ve daha sonra bunu daha özel sınıflara (yani alt sınıflara) genişletmemize olanak tanır.

Aynı türdeki nesneleri modellemek için bir sınıf kullanırız.

Farklı sınıflar, diğer sınıflar tarafından paylaşılabilen bir sınıfta genelleştirilebilen bazı ortak özelliklere ve davranışlara sahip olabilir.

Genelleştirilmiş sınıfı genişleten özel bir sınıf tanımlayabilirsiniz.

Özel sınıflar, özellikleri ve metotları genel sınıftan devralır.

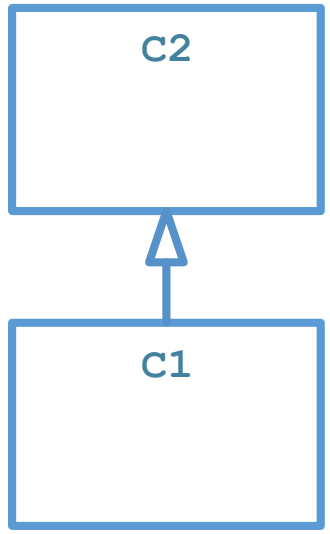
Üst sınıflar ve alt sınıflar (Superclasses and subclasses)

Java terminolojisinde, başka bir C2 sınıfından genişletilmiş bir C1 sınıfı bir alt sınıf olarak adlandırılır ve C2 bir üst sınıf olarak adlandırılır.

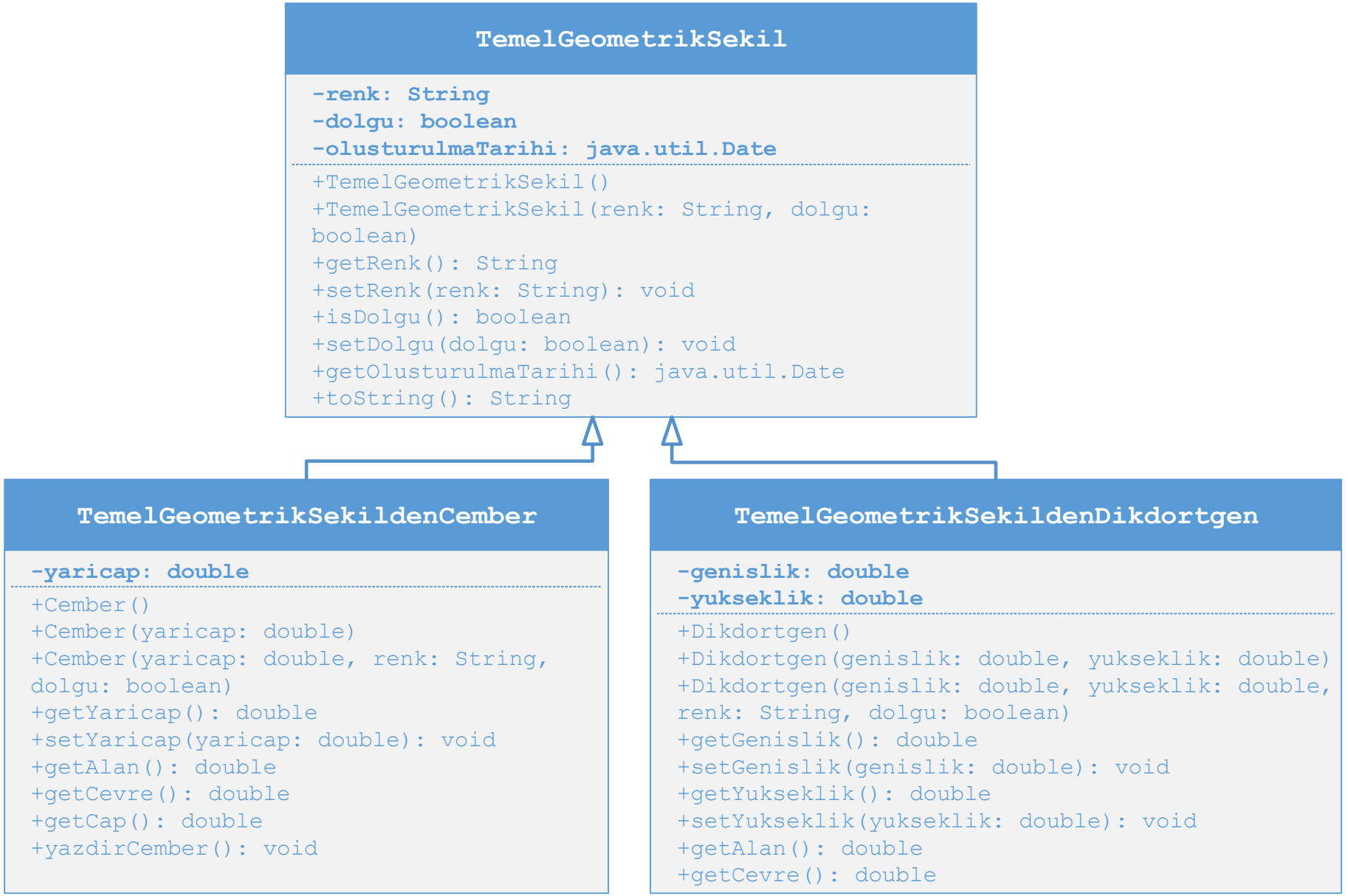
Bir üst sınıfa ayrıca bir ebeveyn (ana) sınıf veya bir temel sınıf

ve bir alt sınıfa bir çocuk sınıf, bir genişletilmiş sınıf veya bir türetilmiş sınıf olarak da başvurulur.

Bir alt sınıf, erişilebilir veri alanlarını ve metotlarını üst sınıfından devralır ve ayrıca yeni veri alanları ve metotlar ekleyebilir.

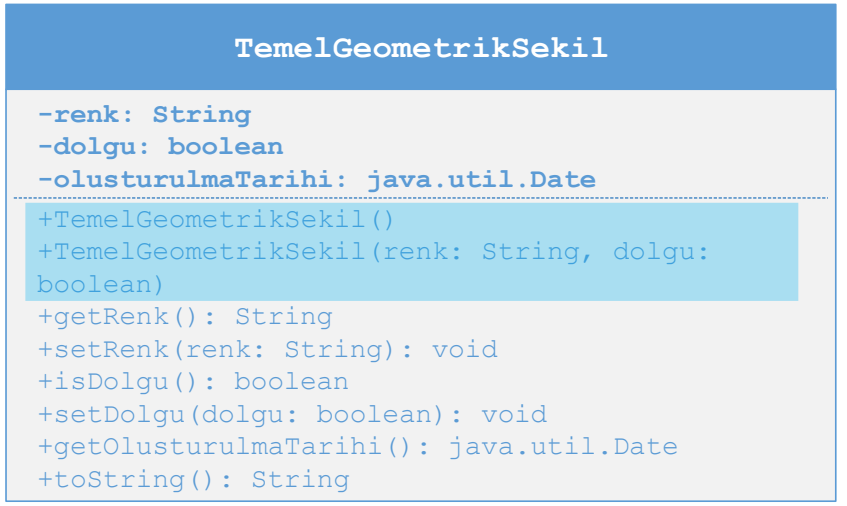


Üst sınıflar ve alt sınıflar (Superclasses and subclasses)

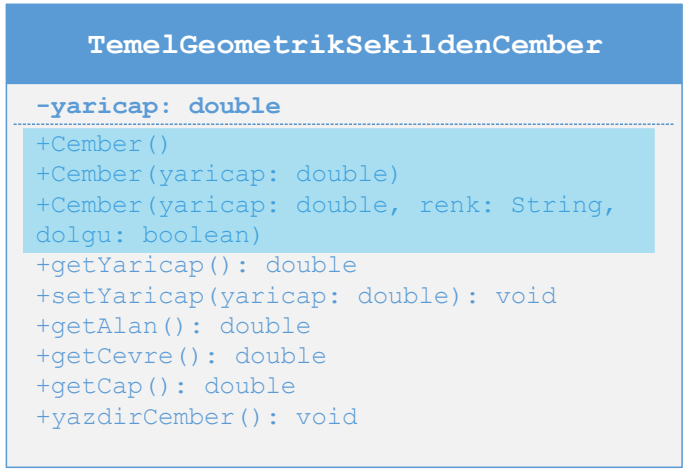


TemelGeometrikSekil.java

```java
public class TemelGeometrikSekil {
    private String renk = "beyaz";
    private boolean dolgu;
    private java.util.Date olusturulmaTarihi;

    /** Varsayilan bir geometrik nesne olustur */
    public TemelGeometrikSekil() {
        olusturulmaTarihi = new java.util.Date();
    }

    /** Belirtilen renk ve dolgu degeri ile
    * varsayilan bir geometrik nesne olustur */
    public TemelGeometrikSekil(String renk,
    boolean dolgu) {
        olusturulmaTarihi = new java.util.Date();
        this.renk = renk;
        this.dolgu = dolgu;
    }

    /** Renk dondur */
    public String getRenk() {
        return renk;
    }

    /** Yeni renk ata */
    public void setRenk(String color) {
        this.renk = renk;
    }

    /** Dolgu dondur. dolgu boolean oldugundan,
    get metodunu is..... seklinde adlandirildi */
    public boolean isDolgu() {
        return dolgu;
    }

    /** Dolgu ata */
    public void setDolgu(boolean dolgu) {
        this.dolgu = dolgu;
    }

    /** OlusturulmaTarihi dondur */
    public java.util.Date getOlusturulmaTarihi() {
        return olusturulmaTarihi;
    }

    /** Bu nesnenin string sunumunu dondur */
    public String toString() {
        return "Olusturulma Tarihi: " + olusturulmaTarihi
        + "\nrenk: " + renk + " ve dolgu: " + dolgu;
    }
}
```



TemelGeometrikSekildenCember.java




```java
public class TemelGeometrikSekildenCember
        extends TemelGeometrikSekil{
    private double yaricap;

    public TemelGeometrikSekildenCember() {
    }

    public TemelGeometrikSekildenCember(double yaricap) {
        this.yaricap = yaricap;
    }

    public TemelGeometrikSekildenCember(double yaricap,
        String renk, boolean dolgu) {
        this.yaricap = yaricap;
        setRenk(renk);
        setDolgu(dolgu);
    }

    /** Yaricap dondur */
    public double getYaricap() {
        return yaricap;
    }

    /** Yeni bir yaricap ata */
    public void setYaricap(double yaricap) {
    this.yaricap = yaricap;
    }
```

```java
    /** Alan dondur */
    public double getAlan() {
        return yaricap * yaricap * Math.PI;
    }

    /** Cap dondur */
    public double getCap() {
        return 2 * yaricap;
    }

    /** Cevre dondur */
    public double getCevre() {
        return 2 * yaricap * Math.PI;
    }

    /** Cember bilgisini yazdir */
    public void yazdirCember() {
        System.out.println("Cember, " + getOlusturulmaTarihi() +
        " tarihinde olusturuldu ve yaricapi " + yaricap + " dir.");
    }
}
```


alt sınıf
subclass
üst sınıf
superclass
public class TemelGeometrikSekildenCember extends TemelGeometrikSekil

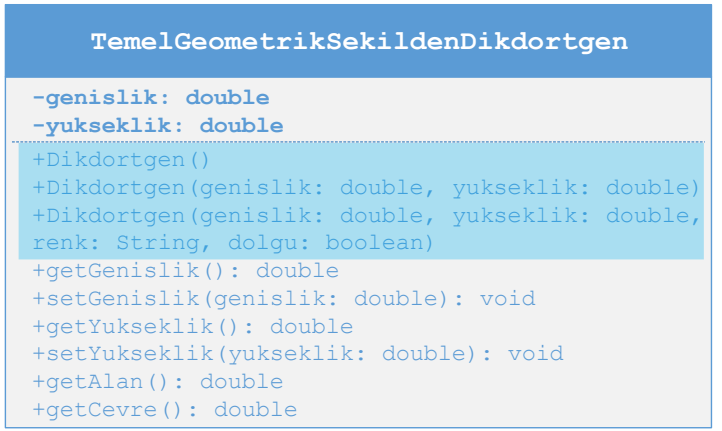

alt sınıf
subclass
üst sınıf
superclass
public class TemelGeometrikSekildenDikdortgen extends TemelGeometrikSekil

private kullanımı

```java
public class TemelGeometrikSekil {
    private String renk = "beyaz";
    private boolean dolgu;
    private java.util.Date olusturulmaTarihi;

    /** Varsayilan bir geometrik nesne olustur */
    public TemelGeometrikSekil() {
        olusturulmaTarihi = new java.util.Date();
    }

    /** Belirtilen renk ve dolgu degeri ile
    * varsayilan bir geometrik nesne olustur */
    public TemelGeometrikSekil(String renk,
    boolean dolgu) {
        olusturulmaTarihi = new java.util.Date();
        this.renk = renk;
        this.dolgu = dolgu;
    }

    /** Renk dondur */
    public String getRenk() {
        return renk;
    }

    /** Yeni renk ata */
    public void setRenk(String color) {
        this.renk = renk;
    }

```

```java
public class TemelGeometrikSekildenCember
        extends TemelGeometrikSekil{
    private double yaricap;

    public TemelGeometrikSekildenCember() {
    }

    public TemelGeometrikSekildenCember(double yaricap) {
        this.yaricap = yaricap;
    }

    public TemelGeometrikSekildenCember(double yaricap,
        String renk, boolean dolgu) {
        this.yaricap = yaricap;
        this.renk;  //yanlış kullanım
        this.dolgu; //yanlış kullanım
    }
    /** Yaricap dondur */
    public double getYaricap() {
        return yaricap;
    }

    /** Yeni bir yaricap ata */
    public void setYaricap(double yaricap) {
    this.yaricap = yaricap;
    }

```

TemelGeometrikSekildenDikdortgen.java




```java
public class TemelGeometrikSekildenDikdortgen
        extends TemelGeometrikSekil {
    private double genislik;
    private double yukseklik;

    public TemelGeometrikSekildenDikdortgen() {
    }

    public TemelGeometrikSekildenDikdortgen(double genislik,
            double yukseklik) {
        this.genislik = genislik;
        this.yukseklik = yukseklik;
    }

    public TemelGeometrikSekildenDikdortgen(double genislik,
            double yukseklik, String renk, boolean dolgu) {
        this.genislik = genislik;
        this.yukseklik = yukseklik;
        setRenk(renk);
        setDolgu(dolgu);
    }

    /** Genislik dondur */
    public double getGenislik() {
        return genislik;
    }

    /** Yeni bir genislik ata */
    public void setGenislik(double genislik) {
        this.genislik = genislik;
    }

    /** Yukseklik dondur */
    public double getYukseklik() {
        return yukseklik;
    }

    /** Yeni bir yukseklik ata */
    public void setYukseklik(double yukseklik) {
        this.yukseklik = yukseklik;
    }

    /** Alan dondur */
    public double getAlan() {
        return genislik * yukseklik;
    }

    /** Cevre dondur */
    public double getCevre() {
        return 2 * (genislik + yukseklik);
    }
}
```

TestCemberDikdortgen.java

```java
public class TestCemberDikdortgen {
    public static void main(String[] args) {
        TemelGeometrikSekildenCember cember = new TemelGeometrikSekildenCember(1);
        System.out.println("Bir cember: " + cember.toString());
        System.out.println("Renk: " + cember.getRenk());
        System.out.println("Yaricap: " + cember.getYaricap());
        System.out.println("Alan: " + cember.getAlan());
        System.out.println("Cap: " + cember.getCap());

        TemelGeometrikSekildenDikdortgen dikdortgen = new TemelGeometrikSekildenDikdortgen(2, 4);
        System.out.println("\nBir dikdortgen: " + dikdortgen.toString());
        System.out.println("Alan: " + dikdortgen.getAlan());
        System.out.println("Cevre: " + dikdortgen.getCevre());
    }
}
```

```
C:\Program Files\Java\jdk-15.0.1\bin\yeni2>java TestCemberDikdortgen
Bir cember: Olusturulma Tarihi: Sun Apr 04 13:21:51 TRT 2021
renk: beyaz ve dolgu: false
Renk: beyaz
Yaricap: 1.0
Alan: 3.141592653589793
Cap: 2.0

Bir dikdortgen: Olusturulma Tarihi: Sun Apr 04 13:21:51 TRT 2021
renk: beyaz ve dolgu: false
Alan: 8.0
Cevre: 12.0
```

super anahtar sözcüğü (keyword)

super anahtar sözcüğü üst sınıfa atıfta bulunur ve üst sınıf metotlarını ve yapıcılarını çağırmak için kullanılabilir.

İki şekilde kullanılabilir:

Bir üst sınıf yapıcısını çağırmak için.

Bir üst sınıf metodunu çağırmak için.

üst sınıf (superclass) yapıcısı çağırma

super() ifadesi üst sınıfın argümansız yapıcısını çağırır.

super(argümanlar) ifadesi argümanlarla eşleşen üst sınıf yapıcısını çağırır.

super() veya super(argümanlar) ifadesi, alt sınıfın kurucusunun ilk ifadesi olmalıdır.

Bu, bir üst sınıf yapıcısını açıkça çağırmanın tek yoludur.

üst sınıf (superclass) yapıcısı çağırma

Üst sınıf kurucusunu çağırmak için super anahtar kelimesini kullanmamız gerekir ve çağrı, yapıcıdaki ilk ifade olmalıdır.

```
1  public class TemelGeometrikSekildenCember
2          extends TemelGeometrikSekil{
3      private double yaricap;
4
5      public TemelGeometrikSekildenCember() {
6      }
7
8      public TemelGeometrikSekildenCember(double yaricap) {
9          this.yaricap = yaricap;
10     }
11
12     public TemelGeometrikSekildenCember(double yaricap,
13         String renk, boolean dolgu) {
14         this.yaricap = yaricap;
15         setRenk(renk);
16         setDolgu(dolgu);
17     }
18
19     /** Yaricap dondur */
20     public double getYaricap() {
21         return yaricap;
22     }
23
24     /** Yeni bir yaricap ata */
25     public void setYaricap(double yaricap) {
26     this.yaricap = yaricap;
27     }
28
```

```
1  public class TemelGeometrikSekildenCember
2          extends TemelGeometrikSekil{
3      private double yaricap;
4
5      public TemelGeometrikSekildenCember() {
6      }
7
8      public TemelGeometrikSekildenCember(double yaricap) {
9          this.yaricap = yaricap;
10     }
11
12     public TemelGeometrikSekildenCember(double yaricap,
13         String renk, boolean dolgu) {
14         super(renk, dolgu);
15         this.yaricap = yaricap;
16     }
17
18     /** Yaricap dondur */
19     public double getYaricap() {
20         return yaricap;
21     }
22
23     /** Yeni bi
24     public void
25     this.yarica
26     }
27
```

Bir alt sınıfta bir üst sınıf kurucusunun adını çağırmak sözdizimi hatasına neden olur.

yapıcı (constructor) zinciri

```
public ClassName() {
  // some statements
}
```

Equivalent

```
public ClassName() {
  super();
  // some statements
}
```

```
public ClassName(double d) {
  // some statements
}
```

Equivalent

```
public ClassName(double d) {
  super();
  // some statements
}
```

Bir yapıcı, aşırı yüklenmiş bir yapıcıyı veya üst sınıf yapıcısını çağırabilir.

Hiçbiri açıkça çağrılmazsa, derleyici otomatik olarak yapıcıya ilk ifade olarak super() koyar.

yapıcı zincirleme (constructor chaining)

Her durumda, bir sınıfın bir örneğini oluşturmak, miras zinciri boyunca tüm üst sınıfların yapıcılarını çağırır.

Bir alt sınıfın bir nesnesini oluştururken, alt sınıf yapıcısı ilk olarak kendi görevlerini gerçekleştirmeden önce üst sınıf yapıcısını çağırır.

Üst sınıf başka bir sınıftan türetilmişse, üst sınıf yapıcısı kendi görevlerini gerçekleştirmeden önce üst sınıf yapıcısını çağırır.

Bu süreç, miras hiyerarşisi boyunca son kurucu çağrılana kadar devam eder. Buna yapıcı zincirleme denir.

Örnek 1: yapıcı zincirleme (constructor chaining)

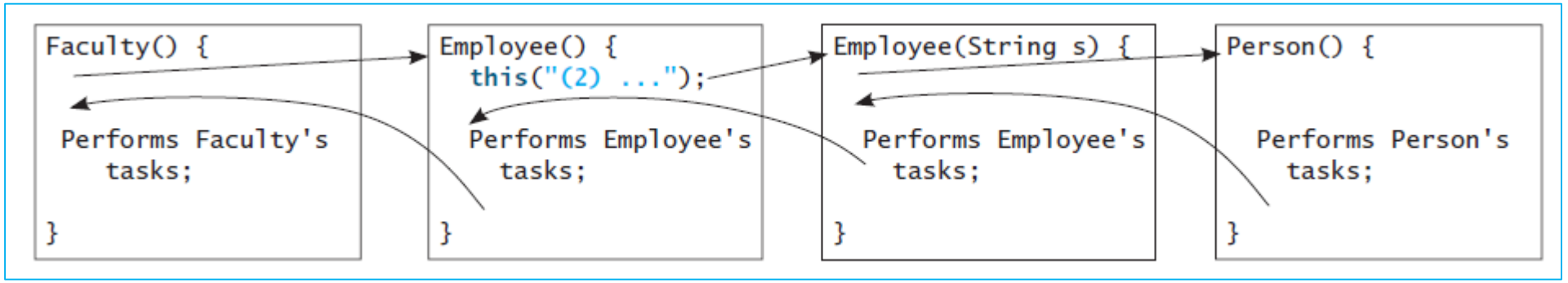
```
1  public class Faculty extends Employee {
2    public static void main(String[] args) {
3      new Faculty();
4    }
5
6    public Faculty() {
7      System.out.println("(4) Performs Faculty's tasks");
8    }
9  }
10
11 class Employee extends Person {
12   public Employee() {
13     this("(2) Invoke Employee's overloaded constructor");
14     System.out.println("(3) Performs Employee's tasks ");
15   }
16
17   public Employee(String s) {
18     System.out.println(s);
19   }
20 }
21
22 class Person {
23   public Person() {
24     System.out.println("(1) Performs Person's tasks");
25   }
26 }
```

```
(1) Performs Person's tasks
(2) Invoke Employee's overloaded constructor
(3) Performs Employee's tasks
(4) Performs Faculty's tasks
```

Örnek 1: yapıcı zincirleme (constructor chaining)

```
1  public class Faculty extends Employee {
2    public static void main(String[] args) {
3      new Faculty();
4    }
5
6    public Faculty() {
7      System.out.println("(4) Performs Faculty's tasks");
8    }
9  }
10
11 class Employee extends Person {
12   public Employee() {
13     this("(2) Invoke Employee's overloaded constructor");
14     System.out.println("(3) Performs Employee's tasks ");
15   }
16
17   public Employee(String s) {
18     System.out.println(s);
19   }
20 }
21
22 class Person {
23   public Person() {
24     System.out.println("(1) Performs Person's tasks");
25   }
26 }
```

```
(1) Performs Person's tasks
(2) Invoke Employee's overloaded constructor
(3) Performs Employee's tasks
(4) Performs Faculty's tasks
```



üst sınıf (superclass) metodu çağırma

super anahtar sözcüğü, üst sınıftaki yapıcıdan başka bir metoda başvurmak için de kullanılabilir.

Sözdizimi şöyledir:

super.metot(parametre);

```
/** Cember bilgisini yazdir */
public void yazdirCember() {
    System.out.println("Cember, " + super.getOlusturulmaTarihi() +
    " tarihinde olusturuldu ve yaricapi " + yaricap + " dir.");
}
```

public, protected, default, private

TABLE 11.2 Data and Methods Visibility

Modifier on members in a class	*Accessed from the same class*	*Accessed from the same package*	*Accessed from a subclass in a different package*	*Accessed from a different package*
public	✓	✓	✓	✓
protected	✓	✓	✓	–
default (no modifier)	✓	✓	–	–
private	✓	–	–	–

Visibility increases

private, default (no modifier), protected, public

package, public, protected, default, private

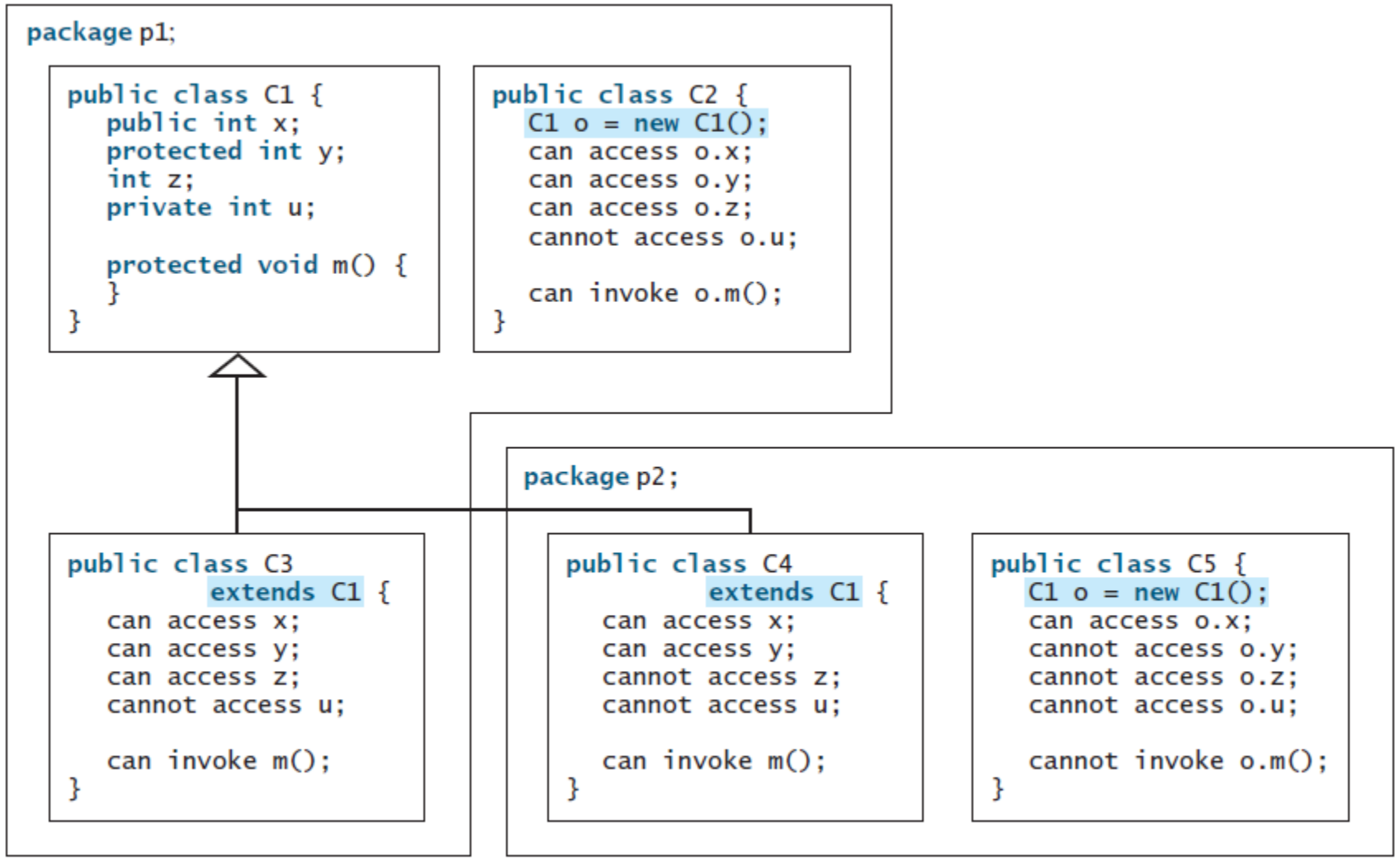


FIGURE 11.5 Visibility modifiers are used to control how data and methods are accessed.

Ödev 1:

Aşağıdaki programların çıktısı nedir?

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overrides the method in B
  public void p(double i) {
    System.out.println(i);
  }
}
```

(a)

```
public class Test {
  public static void main(String[] args) {
    A a = new A();
    a.p(10);
    a.p(10.0);
  }
}

class B {
  public void p(double i) {
    System.out.println(i * 2);
  }
}

class A extends B {
  // This method overloads the method in B
  public void p(int i) {
    System.out.println(i);
  }
}
```

(b)