



# Chapter 9

# Object-Oriented

# Programming: Inheritance

Java™ How to Program, 10/e



# Amaçlar

- Bu bölümde kalıtımın (inheritance) nasıl kullanıldığını ve var olan sınıfları kullanarak yeni sınıflar geliştirmeyi öğreneceksiniz.
- Superclass ve subclass kavramlarını ve bunların birbiriyle ilişkilerini göreceksiniz.
- **extends** anahtar kelimesini kullanarak bir sınıfın özellikleri ve davranışları kalıtım ile başka bir sınıfa almayı öğreneceksiniz.
- Protected access modifierı (erişim belirleyicisi) kullanarak alt sınıfların bu modifiere sahip özellikleri kullanabileceklerini göreceksiniz.
- Üst sınıfın (superclass) üyelerine **super** anahtar kelimesi ile erişebilmek.
- Kalıtım kullanıldığından constructor nasıl çalışıyor bunun incelenmesi.
- Doğrudan ya da dolaylı olarak tüm sınıfların superclassı olan *Object* sınıfının metodlarını göreceğiz.



---

## 9.1 Introduction

## 9.2 Superclasses and Subclasses

## 9.3 **protected** Members

## 9.4 Relationship Between Superclasses and Subclasses

9.4.1 Creating and Using a `CommissionEmployee` Class

9.4.2 Creating and Using a `BasePlusCommissionEmployee` Class

9.4.3 Creating a `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy

9.4.4 `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy Using **protected** Instance Variables

9.4.5 `CommissionEmployee`–`BasePlusCommissionEmployee` Inheritance Hierarchy Using **private** Instance Variables

## 9.5 Constructors in Subclasses

## 9.6 Class Object

## 9.7 (Optional) GUI and Graphics Case Study: Displaying Text and Images Using **Labels**

## 9.8 Wrap-Up

---



## 9.1 Giriş

### ► Inheritance (Kalıtım)

- Yeni bir sınıf, var olan bir sınıfın üyelerini yeni ve değiştirilmiş etmenlerle donatarak oluşturulmaktadır.
- Daha önceden test edilmiş var olan metot ve özelliklerin kullanılması program geliştirilme sürecinde zaman kazandırır.
- Sistemin uygulanması ve bakımının verimli olma olasılığını arttırır.



## 9.1 Giriş(Dvm.)

- ▶ Bir sınıf yaratıldığında, tamamıyla yeni üyeleri tanımlamak yerine varolan sınıfın elemanlarını kullanarak yeni bir sınıf tasarılayabilirsiniz.
    - Halihazırda varolan sınıf **superclass** (üst sınıf)
    - Yeni oluşacak sınıf ise **subclass** (alt sınıf) olmaktadır.
  - ▶ Subclass ilerde başka sınıfların superclassı olabilir
  - ▶ Bir subclass kendi fieldlerini (alanlarını) ve metodlarını ekleyebilir.
  - ▶ Bir subclass kendi üst sınıfından daha spesifiktir ve daha spesifik bir grup nesneye hitap eder.
  - ▶ Subclass üst sınıfının davranışlarını ve özelliklerini gösterir üstüne de kendisi için yeni davranışlar ve özellikler ekler.
- Inheritance'a specialization da denilmesinin nedeni budur..



## 9.1 Giriş (Dvm.)

- ▶ Java sınıfı hiyerarşisi Object sınıfıyla başlar (paket java.lang'da)
- ▶ Java'daki her sınıf doğrudan veya dolaylı olarak Object'i genişletir (veya “ondan miras alır”).
- ▶ Java, her sınıfın tam olarak bir doğrudan üst sınıfından türetildiği tekil kalıtımı (single inheritance'ı) destekler.



## 9.1 Giriş (Dvm.)

- ▶ *is-a relationship* ve *has-a relationship* farklı ilişkiler gösterilmektedir.
- ▶ *Is-a* kalıtımı (inheritance) gösterir.
  - *is-a* ilişkisinde subclassın bir nesnesi üst sınıfın kendisi gibi davranır.
- *Has-a* bileşimi temsil eder.
  - *has-a* ilişkisinde, bir nesne bakşa nesnelerin referanslarını üye olarak içerir.

## 9.2 Superclasslar ve Subclasslar

- ▶ Şekil 9.1'de super sınıfların ve alt sınıfların çeşitli örnekleri görülmektedir.
  - Superclasslar "daha genel" ve alt sınıflar "daha özeldir."
- ▶ Tüm alt sınıfın tüm nesneleri üst sınıfın bir nesnesidir ve bir superclass birçok alt sınıfa sahip olabilir.

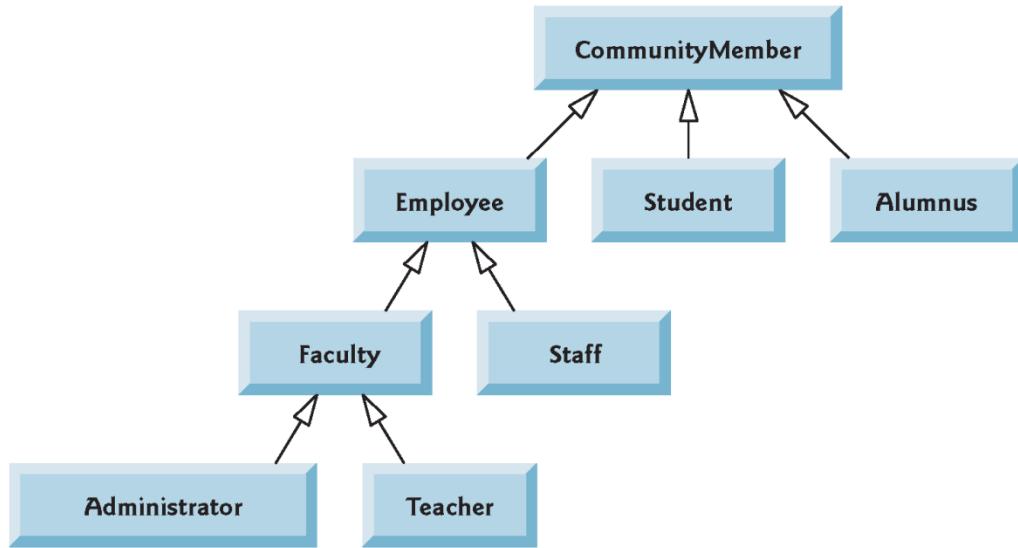
Superclass	Subclasses
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
BankAccount	CheckingAccount, SavingsAccount

**Fig. 9.1** | Inheritance examples.

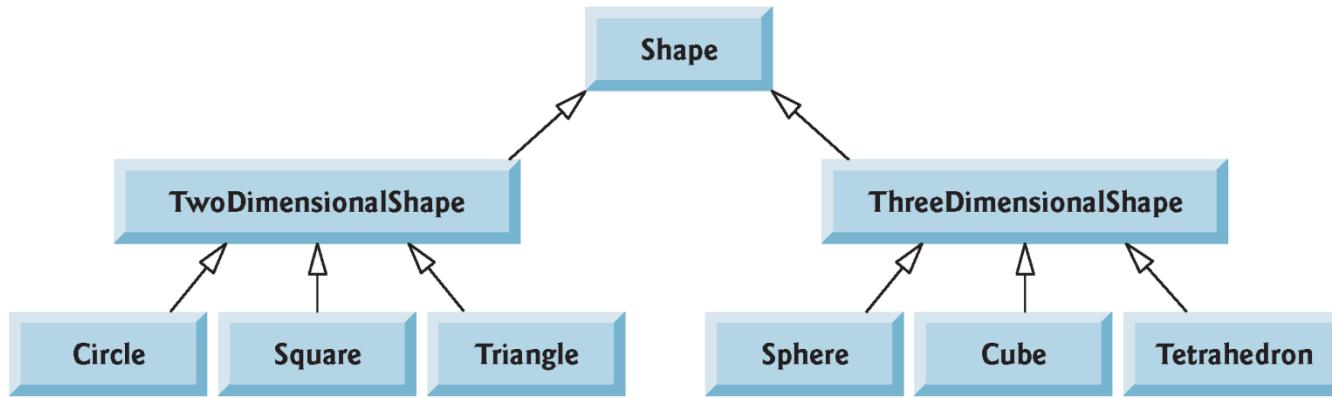


## 9.2 Superclasslar ve Subclasslar(Dvm.)

- ▶ Hiyerarşideki her ok, bir-is a relation'ını temsil eder.
- ▶ Sınıf hiyerarşisinde yukarıya giden okları takip edin
  - “Employee *bir* CommunityMember”’dır
  - “a Teacher *bir* Faculty member.” ‘dır
- CommunityMember, Employee, Student ve Alumnus'un doğrudan üst sınıfıdır ve şemadaki diğer tüm sınıfların dolaylı bir üst sınıfıdır.
- ▶ En alttan başlayarak, okları takip edebilir ve en yüksek süper-sınıfa kadar olan is-a ilişkisini uygulayabilirsiniz.



**Fig. 9.2** | Inheritance hierarchy UML class diagram for university  
CommunityMembers.



**Fig. 9.3** | Inheritance hierarchy UML class diagram for Shapes.



## 9.2 Superclasslar ve Subclasslar (Dvm.)

- ▶ Sınıflar arasında kalıtım ilişkisi illa yoktur.
- ▶ *Has-a* ilişkisi
  - Bir sınıf içerisinde başka sınıf nesnesi üye olarak eklendiğinde oluşur.
  - Örnek: Verilen Employee, BirthDate ve TelephoneNumber sınıflarında, Employee bir BirthDate dir ya da Employee bir TelephoneNumber dir demek yanlıştır .
  - Ancak, bir Employee BirthDate' e sahiptir, ve yine bir Employee TelephoneNumber' e sahiptir.



## 9.2 Superclasslar ve Subclasslar (Dvm)

- ▶ Ortak bir süpersınıfı extend eden (ondan kalıtım alan) tüm sınıfların nesneleri, bu üst sınıfın nesneleri olarak ele alınabilir..
  - Ortaklık, üst sınıfın üyelerinde ifade edilir.
- ▶ Kalıtım Konusu (Inheritance issue)
  - Bir alt sınıf, ihtiyaç duymadığı veya sahip olmaması gereken metotları kalıtımıla alabilir.
  - Bir üst sınıf metodu bir alt sınıf için uygun olduğunda bile, bu alt sınıfın genellikle metodun özelleştirilmiş bir sürümüne ihtiyacı vardır.
  - Alt sınıf, süper sınıf yöntemini uygun bir metot ile geçersiz kılabilir (yeniden yazabilir) **override**.



## 9.3 **protected** Üyeler (Members)

- ▶ Bir sınıfın **public** üyelerine, programın o sınıfın bir nesnesine veya alt sınıflarından birinin bir referansına sahip olduğu her yerde erişilebilir.
- ▶ Bir sınıfın **private** üyelerine yalnızca sınıfın kendisinde erişilebilir.
- ▶ **protected** erişim, **public** ve **private** arasındaki orta düzeyde bir erişim düzeyiyidir.
  - Üst sınıfın **protected** üyelerine bu süper sınıfın üyeleri tarafından, alt sınıflarının üyeleri ve aynı paketteki diğer sınıfların üyeleri tarafından erişilebilir.
    - Korumalı üyeler de paket erişimine sahip.
  - Tüm **public** ve **protected** süper sınıf üyeleri, alt sınıfın üyesi oldukları orijinal erişim belirleyicilerini korurlar.



## 9.3 **protected** Üyeler (Members) (Dvm.)

- ▶ Üst sınıfın **private** üyeleri alt sınıflarından gizlenir.
  - Yalnızca süper sınıfın miras aldığı **public** veya **protected** yöntemlerle erişilebilirler.
- ▶ Alt sınıf metodları, üye isimlerini kullanarak süper sınıfın miras aldığı **public** ve **protected** üyelerine erişebilir.
- ▶ Bir alt sınıf metodu, devralınan bir üst sınıf metodunu geçersiz kıalarsa (override ederse), metodun üst sınıf sürümüne süper sınıf yöntemi adından önce **super** anahtar ve bir nokta ( . ) ayırıcısı kullanılarak alt sınıftan erişilebilir.



## 9.4 Alt Sınıflar ve Üst Sınıflar Arasındaki İlişki

- ▶ Bir şirketin bordro uygulamasında çalışan kişilerin çalışma türlerini içeren kalitim hiyerarşisi üzerinde çalışacağız.
- ▶ Komisyon çalışanlarına (*Commission employees*) satışlarının belirli bir yüzdesi ödeniyor.
- ▶ Baz maaşlı komisyon çalışanları (*Base-salaried commission employees*), bir taban maaşı ve satışlarının bir yüzdesini alırlar.



## 9.4.1 CommissionEmployee Sınıfını Oluşturma ve Kullanma

- ▶ **CommissionEmployee** (Fig. 9.4) sınıfı, **Object** (from package **java.lang**) sınıfını **extend** eder.
  - **CommissionEmployee** sınıfı **Object** metodlarını extend eder.
  - Yeni bir sınıfın hangi sınıfın extend edeceğini açıkça belirtmezseniz, sınıf, **Object** üstü kapalı olarak genişletir.



```
1 // Fig. 9.4: CommissionEmployee.java
2 // CommissionEmployee class represents an employee paid a
3 // percentage of gross sales.
4 public class CommissionEmployee extends Object
5 {
6     private final String firstName;
7     private final String lastName;
8     private final String socialSecurityNumber;
9     private double grossSales; // gross weekly sales
10    private double commissionRate; // commission percentage
11
12    // five-argument constructor
13    public CommissionEmployee(String firstName, String lastName,
14        String socialSecurityNumber, double grossSales,
15        double commissionRate)
16    {
17        // implicit call to Object's default constructor occurs here
18
19        // if grossSales is invalid throw exception
20        if (grossSales < 0.0)
21            throw new IllegalArgumentException(
22                "Gross sales must be >= 0.0");
23    }
```

---

**Fig. 9.4** | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part I of 5.)



---

```
24     // if commissionRate is invalid throw exception
25     if (commissionRate <= 0.0 || commissionRate >= 1.0)
26         throw new IllegalArgumentException(
27             "Commission rate must be > 0.0 and < 1.0");
28
29     this.firstName = firstName;
30     this.lastName = lastName;
31     this.socialSecurityNumber = socialSecurityNumber;
32     this.grossSales = grossSales;
33     this.commissionRate = commissionRate;
34 } // end constructor
35
36 // return first name
37 public String getFirstName()
38 {
39     return firstName;
40 }
41
42 // return last name
43 public String getLastName()
44 {
45     return lastName;
46 }
```

---

**Fig. 9.4** | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part 2 of 5.)



---

```
47
48     // return social security number
49     public String getSocialSecurityNumber()
50     {
51         return socialSecurityNumber;
52     }
53
54     // set gross sales amount
55     public void setGrossSales(double grossSales)
56     {
57         if (grossSales < 0.0)
58             throw new IllegalArgumentException(
59                 "Gross sales must be >= 0.0");
60
61         this.grossSales = grossSales;
62     }
63
64     // return gross sales amount
65     public double getGrossSales()
66     {
67         return grossSales;
68     }
69
```

---

**Fig. 9.4** | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part 3 of 5.)



---

```
70 // set commission rate
71 public void setCommissionRate(double commissionRate)
72 {
73     if (commissionRate <= 0.0 || commissionRate >= 1.0)
74         throw new IllegalArgumentException(
75             "Commission rate must be > 0.0 and < 1.0");
76
77     this.commissionRate = commissionRate;
78 }
79
80 // return commission rate
81 public double getCommissionRate()
82 {
83     return commissionRate;
84 }
85
```

---

**Fig. 9.4** | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part 4 of 5.)



```
86 // calculate earnings
87 public double earnings()
88 {
89     return commissionRate * grossSales;
90 }
91
92 // return String representation of CommissionEmployee object
93 @Override // indicates that this method overrides a superclass method
94 public String toString()
95 {
96     return String.format("%s: %s %s%n%s: %s%n%s: %.2f%n%s: %.2f",
97             "commission employee", firstName, lastName,
98             "social security number", socialSecurityNumber,
99             "gross sales", grossSales,
100            "commission rate", commissionRate);
101 }
102 } // end class CommissionEmployee
```

**Fig. 9.4** | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part 5 of 5.)



## 9.4.1 CommissionEmployee Sınıfını Oluşturma ve Kullanma (Dvm.)

- ▶ Constructorlar kalıtımıla alınmazlar.
- ▶ Bir alt sınıf kurucusunun (constructor) ilk görevi, doğrudan üst sınıfın constructor'ını açıkça veya örtülü olarak çağrırmaktır.
  - Üst sınıfın devralınan örnek değişkenlerin doğru şekilde başlatıldığından emin olmak için.
  - Kod, üst sınıf yapıcısına açık bir çağrı içermiyorsa, Java, üst sınıfın varsayılan veya no-argüman kurucusunu üstü kapalı olarak çağrıır.
  - ▶ Bir sınıfın varsayılan kurucusu, süper sınıfın varsayılan veya no-argüman kurucusunu çağrıır.

## 9.4.1 CommissionEmployee Sınıfını Oluşturma ve Kullanma (Dvm.)

- ▶ **toString**, her nesnenin **Object** sınıfından doğrudan veya dolaylı olarak devraldığı yöntemlerden biridir.
  - Bir nesneyi temsil eden bir **String** döndürür.
  - Bir nesne bir **String** temsiline dönüştürülmesi gerekiğinde üstü kapalı olarak çağrılır.
- ▶ **Object**'s sınıfının **toString** yöntemi, nesnenin sınıfının adını içeren bir **String** döndürür.
- ▶ Bu, uygun bir **String** temsilini belirtmek için bir alt sınıf tarafından override edilebilen bir yer tutucudur.



## 9.4.1 CommissionEmployee Sınıfını Oluşturma ve Kullanma (Dvm.)

- ▶ Bir alt sınıf metodunu geçersiz kılmak için, bir alt sınıfın, üst sınıf metodu ile aynı imzaya sahip bir yöntemi bildirmesi gereklidir.
- ▶ **@Override bildirimini**
  - Bir metodun, aynı imzaya sahip bir üst sınıf metodunu geçersiz kıldığını belirtir.
  - Aynı imzaya sahip olmazsa derleme hatası olur.



# Hata

Override notunun yazılmış yazılmasına size kalmasına rağmen, imzaların düzgün verilmesini garantilemek için önemlidir.



# Hata

Bir metodu daha kısıtlı bir erişim belirleyici ile override etmek derleme hatasına neden olmaktadır. Örneğin süper sınıfta public olan bir metod alt sınıfta private ya da protected olamamaktadır.



```
1 // Fig. 9.5: CommissionEmployeeTest.java
2 // CommissionEmployee class test program.
3
4 public class CommissionEmployeeTest
5 {
6     public static void main(String[] args)
7     {
8         // instantiate CommissionEmployee object
9         CommissionEmployee employee = new CommissionEmployee(
10             "Sue", "Jones", "222-22-2222", 10000, .06);
11
12     // get commission employee data
13     System.out.println(
14         "Employee information obtained by get methods:");
15     System.out.printf("%n%s %s%n", "First name is",
16         employee.getFirstName());
17     System.out.printf("%s %s%n", "Last name is",
18         employee.getLastName());
19     System.out.printf("%s %s%n", "Social security number is",
20         employee.getSocialSecurityNumber());
21     System.out.printf("%s %.2f%n", "Gross sales is",
22         employee.getGrossSales());
23     System.out.printf("%s %.2f%n", "Commission rate is",
24         employee.getCommissionRate());
```

**Fig. 9.5** | CommissionEmployee class test program. (Part I of 2.)



```
25  
26     employee.setGrossSales(5000);  
27     employee.setCommissionRate(.1);  
28  
29     System.out.printf("%n%s:%n%n%s%n",  
30                         "Updated employee information obtained by toString", employee);  
31 } // end main  
32 } // end class CommissionEmployeeTest
```

Employee information obtained by get methods:

First name is Sue  
Last name is Jones  
Social security number is 222-22-2222  
Gross sales is 10000.00  
Commission rate is 0.06

Updated employee information obtained by toString:

commission employee: Sue Jones  
social security number: 222-22-2222  
gross sales: 5000.00  
commission rate: 0.10

**Fig. 9.5** | CommissionEmployee class test program. (Part 2 of 2.)



## 9.4.2 BasePlus-CommissionEmployee Sınıfının Oluşturulması ve Kullanılması

- ▶ BasePlusCommissionEmployee sınıfı(Şekil 9.6) bir ad, soyadı, sosyal güvenlik numarası, brüt satış tutarı, komisyon oranı ve taban maaşı içerir.
  - Baz maaş hariç hepsi, CommissionEmployee sınıfı ile ortaktır.
- ▶ Class BasePlusCommissionEmployee'nin public metotları arasında bir kurucu ve kazanç hesaplama metodu , toString ve her bir değişken için get ve set metotları bulunmaktadır.
  - Çoğu CommissionEmployee ile ortaktır .



```
1 // Fig. 9.6: BasePlusCommissionEmployee.java
2 // BasePlusCommissionEmployee class represents an employee who receives
3 // a base salary in addition to commission.
4
5 public class BasePlusCommissionEmployee
{
6
7     private final String firstName;
8     private final String lastName;
9     private final String socialSecurityNumber;
10    private double grossSales; // gross weekly sales
11    private double commissionRate; // commission percentage
12    private double baseSalary; // base salary per week
13
14    // six-argument constructor
15    public BasePlusCommissionEmployee(String firstName, String lastName,
16        String socialSecurityNumber, double grossSales,
17        double commissionRate, double baseSalary)
18    {
19        // implicit call to Object's default constructor occurs here
20
21        // if grossSales is invalid throw exception
22        if (grossSales < 0.0)
23            throw new IllegalArgumentException(
24                "Gross sales must be >= 0.0");
```

---

**Fig. 9.6** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part I of 6.)



```
25
26     // if commissionRate is invalid throw exception
27     if (commissionRate <= 0.0 || commissionRate >= 1.0)
28         throw new IllegalArgumentException(
29             "Commission rate must be > 0.0 and < 1.0");
30
31     // if baseSalary is invalid throw exception
32     if (baseSalary < 0.0)
33         throw new IllegalArgumentException(
34             "Base salary must be >= 0.0");
35
36     this.firstName = firstName;
37     this.lastName = lastName;
38     this.socialSecurityNumber = socialSecurityNumber;
39     this.grossSales = grossSales;
40     this.commissionRate = commissionRate;
41     this.baseSalary = baseSalary;
42 } // end constructor
43
44 // return first name
45 public String getFirstName()
46 {
47     return firstName;
48 }
```

**Fig. 9.6** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 2 of 6.)



---

```
49
50     // return last name
51     public String getLastName()
52     {
53         return lastName;
54     }
55
56     // return social security number
57     public String getSocialSecurityNumber()
58     {
59         return socialSecurityNumber;
60     }
61
62     // set gross sales amount
63     public void setGrossSales(double grossSales)
64     {
65         if (grossSales < 0.0)
66             throw new IllegalArgumentException(
67                 "Gross sales must be >= 0.0");
68
69         this.grossSales = grossSales;
70     }
71
```

---

**Fig. 9.6** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 3 of 6.)

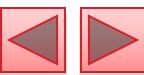


---

```
72     // return gross sales amount
73     public double getGrossSales()
74     {
75         return grossSales;
76     }
77
78     // set commission rate
79     public void setCommissionRate(double commissionRate)
80     {
81         if (commissionRate <= 0.0 || commissionRate >= 1.0)
82             throw new IllegalArgumentException(
83                 "Commission rate must be > 0.0 and < 1.0");
84
85         this.commissionRate = commissionRate;
86     }
87
88     // return commission rate
89     public double getCommissionRate()
90     {
91         return commissionRate;
92     }
93
```

---

**Fig. 9.6** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 4 of 6.)



```
94 // set base salary
95 public void setBaseSalary(double baseSalary)
96 {
97     if (baseSalary < 0.0)
98         throw new IllegalArgumentException(
99             "Base salary must be >= 0.0");
100
101    this.baseSalary = baseSalary;
102 }
103
104 // return base salary
105 public double getBaseSalary()
106 {
107     return baseSalary;
108 }
109
110 // calculate earnings
111 public double earnings()
112 {
113     return baseSalary + (commissionRate * grossSales);
114 }
115
```

**Fig. 9.6** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 5 of 6.)



---

```
116 // return String representation of BasePlusCommissionEmployee
117 @Override
118 public String toString()
119 {
120     return String.format(
121         "%s: %s %s%n%s: %s%n%s: %.2f%n%s: %.2f",
122         "base-salaried commission employee", firstName, lastName,
123         "social security number", socialSecurityNumber,
124         "gross sales", grossSales, "commission rate", commissionRate,
125         "base salary", baseSalary);
126 }
127 } // end class BasePlusCommissionEmployee
```

---

**Fig. 9.6** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 6 of 6.)

## 9.4.2 BasePlus-CommissionEmployee

### Sınıfının Oluşturulması ve Kullanılması (Dvm)

- ▶ Class `BasePlusCommissionEmployee` does *not* specify “`extends Object`”
  - Üstü kapalı olarak `Object`'ı extend eder.
- ▶ `BasePlusCommissionEmployee`'s kurucusu `Object`'s default kurucusunu *üstü kapalı olarak* çağrıır.



---

```
1 // Fig. 9.7: BasePlusCommissionEmployeeTest.java
2 // BasePlusCommissionEmployee test program.
3
4 public class BasePlusCommissionEmployeeTest
5 {
6     public static void main(String[] args)
7     {
8         // instantiate BasePlusCommissionEmployee object
9         BasePlusCommissionEmployee employee =
10            new BasePlusCommissionEmployee(
11                "Bob", "Lewis", "333-33-3333", 5000, .04, 300);
12
13     // get base-salaried commission employee data
14     System.out.println(
15         "Employee information obtained by get methods:%n");
16     System.out.printf("%s %s%n", "First name is",
17         employee.getFirstName());
18     System.out.printf("%s %s%n", "Last name is",
19         employee.getLastName());
20     System.out.printf("%s %s%n", "Social security number is",
21         employee.getSocialSecurityNumber());
22     System.out.printf("%s %.2f%n", "Gross sales is",
23         employee.getGrossSales());
```

---

**Fig. 9.7** | BasePlusCommissionEmployee test program. (Part I of 3.)



---

```
24     System.out.printf("%s %.2f%n", "Commission rate is",
25         employee.getCommissionRate());
26     System.out.printf("%s %.2f%n", "Base salary is",
27         employee.getBaseSalary());
28
29     employee.setBaseSalary(1000);
30
31     System.out.printf("%n%s:%n%n%s%n",
32         "Updated employee information obtained by toString",
33         employee.toString());
34 } // end main
35 } // end class BasePlusCommissionEmployeeTest
```

---

**Fig. 9.7** | BasePlusCommissionEmployee test program. (Part 2 of 3.)



Employee information obtained by get methods:

```
First name is Bob  
Last name is Lewis  
Social security number is 333-33-3333  
Gross sales is 5000.00  
Commission rate is 0.04  
Base salary is 300.00
```

Updated employee information obtained by toString:

```
base-salaried commission employee: Bob Lewis  
social security number: 333-33-3333  
gross sales: 5000.00  
commission rate: 0.04  
base salary: 1000.00
```

**Fig. 9.7** | BasePlusCommissionEmployee test program. (Part 3 of 3.)



## 9.4.1 CommissionEmployee Sınıfını Oluşturma ve Kullanma (Dvm.)

- ▶ BasePlusCommissionEmployee's sınıfının kodu, CommissionEmployee sınıfının koduna benzer ve neredeyse aynıdır.
- ▶ private instance değişkenleri `firstName` ve `lastName` ve `setFirstName`, `getFirstName`, `setLastName` ve `getLastName` metotları aynıdır.
  - Sınıfların ikisi de ilgili *get* ve *set* metotlarını içermektedir.
- ▶ Constructorlar neredeyse aynıdır.
  - BasePlusCommissionEmployee'nin kurucusu `baseSalary` instance değişkenini de ilklendirmektedir.
- ▶ `toString` metotları da neredeyse aynıdır
  - BasePlusCommissionEmployee's `toString` metodu `baseSalary` üyesini de çıktı olarak vermektedir.



## 9.4.1 CommissionEmployee Sınıfını Oluşturma ve Kullanma (Dvm.)

- ▶ Biz tam olarak **CommissionEmployee**'nin kodunu kopyalayıp, **BasePlusCommissionEmployee** 'ye yapıştırdık, daha sonra yeni maaşı baz maaşı ve temel maaşı manipüle eden metodlar içerecek şekilde değiştirdik.
  - Bu “kopyala-yapıştır” yaklaşımı genellikle hata eğilimli ve zaman alıcıdır.
  - Aynı kodun kopyalarını bir sisteme dağıtmak, kod bakım sorunları yaratır – Örn: koddaki değişikliklerin birden çok sınıfta yapılması gereklidir.

## 9.4.3 CommissionEmployee– BasePlusCommissionEmployee Kalıtım Hiyerarşisi (Inheritance Hierarchy) Oluşturmak



BasePlusCommissionEmployee sınıfı  
CommissionEmployee sınıfını extend eder (ondan miras alır).

- ▶ Bir BasePlusCommissionEmployee nesnesi *aynı zamanda* CommissionEmployee nesnesidir.
  - Kalıtım ile CommissionEmployee's yeteneklerini alır.
- ▶ BasePlusCommissionEmployee ayrıca baseSalary instance variable'ına sahiptir.
- ▶ Alt sınıf BasePlusCommissionEmployee CommissionEmployee'nin instance variable'larını ve metodlarını miras alır.
  - CommissionEmployee'nin sadece public ve protected değişkenlerini doğrudan ulaşılabilir.



## GÖZLEM

- 1) Bir nesneye dayalı sistemde birbiriyle yakından ilişkili sınıflarla sıkça karşılaşacaksınız.
- 2) Ortak instance variable'ları ve ortak metotları belirleyeceksiniz bunları süper (üst) sınıfta tutacaksınız.
- 3) Ardından alt sınıfları geliştirirken üst sınıfları kullanacaksınız. Bu alt sınıfları üst sınıfta olmayan kendi işleri ve yetenekleri ile özelleştireceksiniz.



## GÖZLEM

Bir alt sınıf oluşturmak bu sınıfın üst sınıfının kodunu etkilemez. Kalıtım üst sınıfın bütünlüğünü korur.



```
1 // Fig. 9.8: BasePlusCommissionEmployee.java
2 // private superclass members cannot be accessed in a subclass.
3
4 public class BasePlusCommissionEmployee extends CommissionEmployee
5 {
6     private double baseSalary; // base salary per week
7
8     // six-argument constructor
9     public BasePlusCommissionEmployee(String firstName, String lastName,
10         String socialSecurityNumber, double grossSales,
11         double commissionRate, double baseSalary)
12     {
13         // explicit call to superclass CommissionEmployee constructor
14         super(firstName, lastName, socialSecurityNumber,
15             grossSales, commissionRate);
16
17         // if baseSalary is invalid throw exception
18         if (baseSalary < 0.0)
19             throw new IllegalArgumentException(
20                 "Base salary must be >= 0.0");
21
22         this.baseSalary = baseSalary;
23     }
```

**Fig. 9.8** | private superclass members cannot be accessed in a subclass. (Part I of 5.)



---

```
24
25     // set base salary
26     public void setBaseSalary(double baseSalary)
27     {
28         if (baseSalary < 0.0)
29             throw new IllegalArgumentException(
30                 "Base salary must be >= 0.0");
31
32         this.baseSalary = baseSalary;
33     }
34
35     // return base salary
36     public double getBaseSalary()
37     {
38         return baseSalary;
39     }
40
41     // calculate earnings
42     @Override
43     public double earnings()
44     {
45         // not allowed: commissionRate and grossSales private in superclass
46         return baseSalary + (commissionRate * grossSales);
47     }
```

---

**Fig. 9.8** | private superclass members cannot be accessed in a subclass. (Part 2 of 5.)

```
48
49 // return String representation of BasePlusCommissionEmployee
50 @Override
51 public String toString()
52 {
53     // not allowed: attempts to access private superclass members
54     return String.format(
55         "%s: %s %s%n%s: %.2f%n%s: %.2f%n%s: %.2f",
56         "base-salaried commission employee", firstName, lastName,
57         "social security number", socialSecurityNumber,
58         "gross sales", grossSales, "commission rate", commissionRate,
59         "base salary", baseSalary);
60 }
61 } // end class BasePlusCommissionEmployee
```

**Fig. 9.8** | private superclass members cannot be accessed in a subclass. (Part 3 of 5.)



```
BasePlusCommissionEmployee.java:46: error: commissionRate has private access  
in CommissionEmployee  
    return baseSalary + (commissionRate * grossSales);  
                           ^  
BasePlusCommissionEmployee.java:46: error: grossSales has private access in  
CommissionEmployee  
    return baseSalary + (commissionRate * grossSales);  
                           ^  
BasePlusCommissionEmployee.java:56: error: firstName has private access in  
CommissionEmployee  
    "base-salaried commission employee", firstName, lastName,  
                           ^
```

**Fig. 9.8** | private superclass members cannot be accessed in a subclass. (Part 4 of 5.)



```
BasePlusCommissionEmployee.java:56: error: lastName has private access in  
CommissionEmployee  
        "base-salaried commission employee", firstName, lastName,  
                           ^  
BasePlusCommissionEmployee.java:57: error: socialSecurityNumber has private  
access in CommissionEmployee  
        "social security number", socialSecurityNumber,  
                           ^  
BasePlusCommissionEmployee.java:58: error: grossSales has private access in  
CommissionEmployee  
        "gross sales", grossSales, "commission rate", commissionRate,  
                           ^  
BasePlusCommissionEmployee.java:58: error: commissionRate has private access  
inCommissionEmployee  
        "gross sales", grossSales, "commission rate", commissionRate,  
                           ^
```

**Fig. 9.8** | private superclass members cannot be accessed in a subclass. (Part 5 of 5.)

## 9.4.3 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hiyerarşisi (Dvm.)

- ▶ Her subclass kurucusu üst sınıfının kurucu metodunu, üstün sınıfın instance variable'larını ilklendirmek için üstü kapalı ya da açık açık çağrırmalıdır.
  - Süper sınıfı kurucusu, sözdizimi — **super** anahtar kelimesini üst sınıf yapıcı argümanlarını içeren bir parantez kümesi izler.
  - Kurucu metodunun ilk satırında olmalıdır.
  - Alt sınıf yapıcısı, üst sınıfın kurucusunu açıkça çağrırmazsa, derleyici, üst sınıfın varsayılan veya no-argüman kurucusuna bir çağrı eklemeyi dener.
  - **CommissionEmployee** sınıfının böyle bir kurucusu yoksa derleyici hata verir.
  - Anahtar kelimesi ile üst sınıfın hiç parametre almayan varsayılan kurucusunu ya da varsayılan constructorını çağrırlırsınız.



## 9.4.3 CommissionEmployee– BasePlusCommissionEmployee Inheritance Hiyerarşisi (Dvm.)



- ▶ Alt sınıf, üst sınıfın **private** örnek değişkenlerine erişmeyi denediğinde derleme hataları oluşur.
- ▶ Bu durumda, üst sınıfın örnek değişkenlerinin değerlerini almak için uygun get yöntemlerini kullanılmalıdır.

## 9.4.4 CommissionEmployee-

BasePlusCommissionEmployee protected

Instance Variables Kullanarak Kalıtım

### Hiyerarşisi

- ▶ Süper sınıf örnek değişkenlerine doğrudan erişebilmek için, bu üyeleri süper sınıfta **protected** olarak tanımlayabiliriz.
- ▶ New **CommissionEmployee** Fig. 9.4'nin 6–10 satırları şu şekilde değiştirilmiştir:

```
protected final String firstName;  
protected final String lastName;  
protected final String socialSecurityNumber;  
protected double grossSales;  
protected double commissionRate;
```

- ▶ **protected** örnek değişkenleri ile alt sınıf üst sınıfın örnek değişkenlerine erişir, ancak alt sınıfı olmayan ve aynı pakette olmayan sınıflar bu değişkenlere doğrudan erişemez



## 9.4.4 CommissionEmployee-

BasePlusCommissionEmployee protected

Instance Variables Kullanarak Kalıtım

### Hiyerarşisi (Dvm)

- ▶ BasePlusCommissionEmployee (Fig. 9.9) sınıfı, CommissionEmployee'nun yeni **protected** instance variable olan halini extend eder.
  - Artı bu değişkenler BasePlusCommissionEmployee sınıfının **protected** üyeleridir.
- ▶ Başka bir sınıf, bu BasePlusCommissionEmployee sınıfını genişletirse, yeni alt sınıf da **protected** üyelerere erişebilir.
- ▶ Şekil 9.9'daki (59 satır) kaynak kodu, Şekil 9.6'dakilerden oldukça kısaltır (127 satır)
  - Most of the functionality is now inherited from CommissionEmployee
  - İşlevlerin çoğu şu anda CommissionEmployee'den miras alınmıştır.
  - Şimdi İşlevin sadece bir kopyası vardır.
  - Kodun bakımı, değiştirilmesi ve hata ayıklaması daha kolaydır - bir CommissionEmployee ilgili kod sadece bu sınıfta bulunur.





```
1 // Fig. 9.9: BasePlusCommissionEmployee.java
2 // BasePlusCommissionEmployee inherits protected instance
3 // variables from CommissionEmployee.
4
5 public class BasePlusCommissionEmployee extends CommissionEmployee
6 {
7     private double baseSalary; // base salary per week
8
9     // six-argument constructor
10    public BasePlusCommissionEmployee(String firstName, String lastName,
11        String socialSecurityNumber, double grossSales,
12        double commissionRate, double baseSalary)
13    {
14        super(firstName, lastName, socialSecurityNumber,
15              grossSales, commissionRate);
16
17        // if baseSalary is invalid throw exception
18        if (baseSalary < 0.0)
19            throw new IllegalArgumentException(
20                "Base salary must be >= 0.0");
21
22        this.baseSalary = baseSalary;
23    }
```

**Fig. 9.9** | BasePlusCommissionEmployee inherits protected instance variables from CommissionEmployee. (Part I of 3.)



---

```
24
25     // set base salary
26     public void setBaseSalary(double baseSalary)
27     {
28         if (baseSalary < 0.0)
29             throw new IllegalArgumentException(
30                 "Base salary must be >= 0.0");
31
32         this.baseSalary = baseSalary;
33     }
34
35     // return base salary
36     public double getBaseSalary()
37     {
38         return baseSalary;
39     }
40
41     // calculate earnings
42     @Override // indicates that this method overrides a superclass method
43     public double earnings()
44     {
45         return baseSalary + (commissionRate * grossSales);
46     }
```

---

**Fig. 9.9** | `BasePlusCommissionEmployee` inherits protected instance variables from `CommissionEmployee`. (Part 2 of 3.)



---

```
47
48     // return String representation of BasePlusCommissionEmployee
49     @Override
50     public String toString()
51     {
52         return String.format(
53             "%s: %s %s%n%s: %s%n%s: %.2f%n%s: %.2f",
54             "base-salaried commission employee", firstName, lastName,
55             "social security number", socialSecurityNumber,
56             "gross sales", grossSales, "commission rate", commissionRate,
57             "base salary", baseSalary);
58     }
59 } // end class BasePlusCommissionEmployee
```

---

**Fig. 9.9** | BasePlusCommissionEmployee inherits protected instance variables from CommissionEmployee. (Part 3 of 3.)

## 9.4.4 CommissionEmployee-

BasePlusCommissionEmployee protected

Instance Variables Kullanarak Kalıtım

### Hiyerarşisi (Dvm)

- ▶ Inheriting **protected** instance variables enables direct access to the variables by subclasses.
- ▶ Coğu durumda, düzgün bir yazılım mühendisliğini teşvik etmek için **private** örnek değişkenlerini kullanmak daha iyidir.
  - Kodun bakımı, değiştirilmesi ve hata ayıklaması daha kolay olacaktır.



## 9.4.4 CommissionEmployee-

BasePlusCommissionEmployee protected

Instance Variables Kullanarak Kalıtım

### Hiyerarşisi (Dvm)

- ▶ **protected** örnek değişkenleri kullanmak birçok potansiyel sorun yaratır.
- ▶ Alt sınıf nesnesi, bir set metodunu kullanmadan doğrudan devralınan bir değişkenin değerini ayarlayabilir.
  - *Bir alt sınıf nesnesi, değişkene geçersiz bir değer atayabilir.*
- ▶ Alt sınıf metodlarının, üst sınıfın veri uygulamasına bağlı olacak şekilde yazılması daha mantıklıdır.
  - Alt sınıflar yalnızca üst sınıf veri uygulamalarına değil, üst sınıf veri hizmetlerine bağlı olmalıdır.





## Hiyerarşisi (Dvm)

- ▶ Üst sınıfın **protected** örnek değişkenlerle, üst sınıf kodu değiştiğinde üst sınıfın tüm alt sınıflarını değiştirmemiz gerekebilir.
  - Böyle bir sınıfın **fragile** ya da **brittle** olduğu söylenir, çünkü üst sınıfın küçük bir değişiklik alt sınıf uygulamasının “bozulması” neden olabilir.
  - Alt sınıflara aynı hizmetleri sunarken, süper sınıf uygulamasını değiştirememiz gerekir.
  - Süper sınıf hizmetleri değişirse, alt sınıflarımızı yeniden düzenlemeliyiz.
  - Bir sınıfın **protected** üyeleri, korunan üyeleri içeren sınıfla aynı paketteki tüm sınıflar tarafından görülebilir - bu her zaman arzu edilmez.



## GÖZLEM

Protected erişim belirleyicisini sadece alt sınıflara ve aynı paketteki diğer sınıflara bir metot verilmesi istendiğinde kullanınız.

Üst sınıfın例中的 örnekte, değişkenlerin private olması alt sınıf kodunu değiştirmeden üst sınıfta uygulamayı değiştiremeyi sağlar.



## 9.4.5 CommissionEmployee–BasePlusCommissionEmployee private Instance Variableler Kullanan Kalıtım Hiyerarşisi

- ▶ CommissionEmployee sınıfı `firstName`, `lastName`, `socialSecurityNumber`, `grossSales` ve `commissionRate` *private* olarak bu değişkenleri manipüle etmek için de `public` metotları oluşturmaktadır.



## 9.4.5 CommissionEmployee–BasePlusCommissionEmployee private Instance

### Variableler Kullanan Kalıtım Hiyerarşisi (Dvm.)

- ▶ **CommissionEmployee** metotları **earnings** ve **toString** sınıfın get metodlarını kullanmaktadır.
- ▶ Verilerin (örneğin, değişken isimleri) iç temsilini değiştirmeye karar verirsek, yalnızca örnek değişkenleri doğrudan işleyen get ve set metod gövdelerinin değişmesi gerekecektir.
- ▶ Bu değişiklikler yalnızca süper sınıf içinde gerçekleşir - alt sınıfa hiçbir değişiklik gerekmektedir.
  - *Bunun gibi değişikliklerin yerelleştirilmesi iyi bir yazılım mühendisliği uygulamasıdır.*
- ▶ Subclass **BasePlusCommissionEmployee** , **Commission-Employee**'in **private** olmayan metodlarını miras alır ve bu yöntemlerle özel süper sınıf üyelerine erişebilir.



```
1 // Fig. 9.10: CommissionEmployee.java
2 // CommissionEmployee class uses methods to manipulate its
3 // private instance variables.
4 public class CommissionEmployee
5 {
6     private final String firstName;
7     private final String lastName;
8     private final String socialSecurityNumber;
9     private double grossSales; // gross weekly sales
10    private double commissionRate; // commission percentage
11
12    // five-argument constructor
13    public CommissionEmployee(String firstName, String lastName,
14        String socialSecurityNumber, double grossSales,
15        double commissionRate)
16    {
17        // implicit call to Object constructor occurs here
18
19        // if grossSales is invalid throw exception
20        if (grossSales < 0.0)
21            throw new IllegalArgumentException(
22                "Gross sales must be >= 0.0");
23    }
```

**Fig. 9.10** | CommissionEmployee class uses methods to manipulate its `private` instance variables. (Part I of 5.)



---

```
24     // if commissionRate is invalid throw exception
25     if (commissionRate <= 0.0 || commissionRate >= 1.0)
26         throw new IllegalArgumentException(
27             "Commission rate must be > 0.0 and < 1.0");
28
29     this.firstName = firstName;
30     this.lastName = lastName;
31     this.socialSecurityNumber = socialSecurityNumber;
32     this.grossSales = grossSales;
33     this.commissionRate = commissionRate;
34 } // end constructor
35
36 // return first name
37 public String getFirstName()
38 {
39     return firstName;
40 }
41
42 // return last name
43 public String getLastName()
44 {
45     return lastName;
46 }
```

---

**Fig. 9.10** | CommissionEmployee class uses methods to manipulate its **private** instance variables. (Part 2 of 5.)



---

```
47
48     // return social security number
49     public String getSocialSecurityNumber()
50     {
51         return socialSecurityNumber;
52     }
53
54     // set gross sales amount
55     public void setGrossSales(double grossSales)
56     {
57         if (grossSales < 0.0)
58             throw new IllegalArgumentException(
59                 "Gross sales must be >= 0.0");
60
61         this.grossSales = grossSales;
62     }
63
64     // return gross sales amount
65     public double getGrossSales()
66     {
67         return grossSales;
68     }
69
```

---

**Fig. 9.10** | CommissionEmployee class uses methods to manipulate its **private** instance variables. (Part 3 of 5.)



---

```
70 // set commission rate
71 public void setCommissionRate(double commissionRate)
72 {
73     if (commissionRate <= 0.0 || commissionRate >= 1.0)
74         throw new IllegalArgumentException(
75             "Commission rate must be > 0.0 and < 1.0");
76
77     this.commissionRate = commissionRate;
78 }
79
80 // return commission rate
81 public double getCommissionRate()
82 {
83     return commissionRate;
84 }
85
86 // calculate earnings
87 public double earnings()
88 {
89     return getCommissionRate() * getGrossSales();
90 }
91
```

---

**Fig. 9.10** | CommissionEmployee class uses methods to manipulate its **private** instance variables. (Part 4 of 5.)



---

```
92     // return String representation of CommissionEmployee object
93     @Override
94     public String toString()
95     {
96         return String.format("%s: %s %s%n%s: %s%n%s: %.2f%n%s: %.2f",
97             "commission employee", getFirstName(), getLastName(),
98             "social security number", getSocialSecurityNumber(),
99             "gross sales", getGrossSales(),
100            "commission rate", getCommissionRate());
101    }
102 } // end class CommissionEmployee
```

---

**Fig. 9.10** | CommissionEmployee class uses methods to manipulate its **private** instance variables. (Part 5 of 5.)



## 9.4.5 CommissionEmployee–BasePlusCommissionEmployee private Instance Variableler Kullanan Kalıtım Hiyerarşisi (Dvm.)

- ▶ Class BasePlusCommissionEmployee bir kaç değişiklik yapılmıştır.
- ▶ `earnings` metodu ve `toString` metodu, her biri kendi üst sınıf sürümlerini çağrıır ve örnek değişkenlerine doğrudan erişmez.



---

```
1 // Fig. 9.11: BasePlusCommissionEmployee.java
2 // BasePlusCommissionEmployee class inherits from CommissionEmployee
3 // and accesses the superclass's private data via inherited
4 // public methods.
5
6 public class BasePlusCommissionEmployee extends CommissionEmployee
7 {
8     private double baseSalary; // base salary per week
9
10    // six-argument constructor
11    public BasePlusCommissionEmployee(String firstName, String lastName,
12        String socialSecurityNumber, double grossSales,
13        double commissionRate, double baseSalary)
14    {
15        super(firstName, lastName, socialSecurityNumber,
16              grossSales, commissionRate);
17    }
```

---

**Fig. 9.11** | BasePlusCommissionEmployee class inherits from CommissionEmployee and accesses the superclass's private data via inherited public methods. (Part I of 3.)



---

```
18     // if baseSalary is invalid throw exception
19     if (baseSalary < 0.0)
20         throw new IllegalArgumentException(
21             "Base salary must be >= 0.0");
22
23     this.baseSalary = baseSalary;
24 }
25
26 // set base salary
27 public void setBaseSalary(double baseSalary)
28 {
29     if (baseSalary < 0.0)
30         throw new IllegalArgumentException(
31             "Base salary must be >= 0.0");
32
33     this.baseSalary = baseSalary;
34 }
35
```

---

**Fig. 9.11** | BasePlusCommissionEmployee class inherits from CommissionEmployee and accesses the superclass's private data via inherited public methods. (Part 2 of 3.)



```
36     // return base salary
37     public double getBaseSalary()
38     {
39         return baseSalary;
40     }
41
42     // calculate earnings
43     @Override
44     public double earnings()
45     {
46         return getBaseSalary() + super.earnings();
47     }
48
49     // return String representation of BasePlusCommissionEmployee
50     @Override
51     public String toString()
52     {
53         return String.format("%s %s%n%s: %.2f", "base-salaried",
54                             super.toString(), "base salary", getBaseSalary());
55     }
56 } // end class BasePlusCommissionEmployee
```

**Fig. 9.11** | BasePlusCommissionEmployee class inherits from CommissionEmployee and accesses the superclass's private data via inherited public methods. (Part 3 of 3.)



## 9.4.5 CommissionEmployee–BasePlusCommissionEmployee private Instance Variableler Kullanan Kalıtım Hiyerarşisi (Dvm.)

- ▶ BasePlusCommissionEmployee's `toString` method overrides class `CommissionEmployee`'s `toString` method.
- ▶ BasePlusCommissionEmployee' `toString` metodu, sınıf `CommissionEmployee`'nin `toString` metodunu geçersiz kılar.
- ▶ Yeni sürüm, `super.toString()` ifadesiyle `CommissionEmployee`'nin `toString` metodunu çağrıarak String temsilinin bir parçasını oluşturur.



## 9.5 Alt sınıfların Kurucuları (Constructors in Subclasses)

- ▶ Bir alt sınıf nesnesini yaratmak, bir yapıçı çağrısı zincirini başlatır.
  - Alt sınıf kurucusu, kendi görevlerini yerine getirmeden önce, doğrudan üst sınıfındaki kuruculardan birini çağırmak için süper üstleri kullanır veya üst sınıfın varsayılan veya bağımsız değişken kurucusunu dolaylı olarak çağırır.
- ▶ Üst sınıf başka bir sınıfından türetilmişse, üst sınıf kurucusu hiyerarşide bir sonraki sınıfın kurucusunu çağırır ve böyle devam eder.
- ▶ Zincirde çağrılan son kurucu her zaman Object yapıcısıdır.



## Software Engineering Observation 9.9

*Java ensures that even if a constructor does not assign a value to an instance variable, the variable is still initialized to its default value (e.g., 0 for primitive numeric types, false for booleans, null for references).*



## 9.6 Object Sınıfı

- ▶ Java'daki tüm sınıflar, doğrudan veya dolaylı olarak Object sınıfından miras alır, böylece Object sınıfının 11 metodu tüm diğer sınıflar tarafından miras alınır.
- ▶ Şekil 9.12, Object yöntemlerini özetler.
- ▶ Her dizi, diziyi kopyalayan override edilmiş `clone` metoduna sahiptir.
  - Dizi, referans nesneleri tutuyorsa, referanslar kopyalanmaz — sığ bir kopya (*shallow*) gerçekleştirilir.



Method	Description
equals	This method compares two objects for equality and returns <code>true</code> if they're equal and <code>false</code> otherwise. The method takes any <code>Object</code> as an argument. When objects of a particular class must be compared for equality, the class should override method <code>equals</code> to compare the <i>contents</i> of the two objects. For the requirements of implementing this method (which include also overriding method <code>hashCode</code> ), refer to the method's documentation at <a href="http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals(java.lang.Object)">docs.oracle.com/javase/7/docs/api/java/lang/Object.html#equals(java.lang.Object)</a> . The default <code>equals</code> implementation uses operator <code>==</code> to determine whether two references <i>refer to the same object</i> in memory. Section 14.3.3 demonstrates class <code>String</code> 's <code>equals</code> method and differentiates between comparing <code>String</code> objects with <code>==</code> and with <code>equals</code> .
hashCode	Hashcodes are <code>int</code> values used for high-speed storage and retrieval of information stored in a data structure that's known as a hashtable (see Section 16.11). This method is also called as part of <code>Object</code> 's default <code>toString</code> method implementation.

**Fig. 9.12** | Object methods. (Part I of 3.)



Method	Description
<code>toString</code>	This method (introduced in Section 9.4.1) returns a <code>String</code> representation of an object. The default implementation of this method returns the package name and class name of the object's class typically followed by a hexadecimal representation of the value returned by the object's <code>hashCode</code> method.
<code>wait</code> , <code>notify</code> , <code>notifyAll</code>	Methods <code>notify</code> , <code>notifyAll</code> and the three overloaded versions of <code>wait</code> are related to multithreading, which is discussed in Chapter 23.
<code>getClass</code>	Every object in Java knows its own type at execution time. Method <code>getClass</code> (used in Sections 10.5 and 12.5) returns an object of class <code>Class</code> (package <code>java.lang</code> ) that contains information about the object's type, such as its class name (returned by <code>Class</code> method <code>getName</code> ).
<code>finalize</code>	This <code>protected</code> method is called by the garbage collector to perform termination housekeeping on an object just before the garbage collector reclaims the object's memory. Recall from Section 8.10 that it's unclear whether, or when, <code>finalize</code> will be called. For this reason, most programmers should avoid method <code>finalize</code> .

**Fig. 9.12** | Object methods. (Part 2 of 3.)



Method	Description
<code>clone</code>	This <b>protected</b> method, which takes no arguments and returns an <code>Object</code> reference, makes a copy of the object on which it's called. The default implementation performs a so-called <b>shallow copy</b> —instance-variable values in one object are copied into another object of the same type. For reference types, only the references are copied. A typical overridden <code>clone</code> method's implementation would perform a <b>deep copy</b> that creates a new object for each reference-type instance variable. <i>Implementing <code>clone</code> correctly is difficult. For this reason, its use is discouraged.</i> Some industry experts suggest that object serialization should be used instead. We discuss object serialization in Chapter 15. Recall from Chapter 7 that arrays are objects. As a result, like all other objects, arrays inherit the members of class <code>Object</code> . Every array has an overridden <code>clone</code> method that copies the array. However, if the array stores references to objects, the objects are not copied—a shallow copy is performed.

**Fig. 9.12** | Object methods. (Part 3 of 3.)