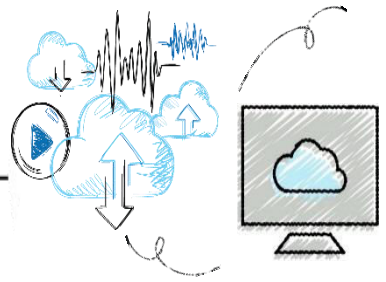
Java Programming, 9e

Chapter 10

Introduction to Inheritance



Objectives

- Describe the concept of inheritance
- Extend classes
- Override superclass methods
- Call constructors during inheritance
- Access superclass methods
- Employ information hiding
- Learn which methods you cannot override

Learning About the Concept of Inheritance

- **Inheritance**
  - A mechanism that enables one class to inherit both the behavior and the attributes of another class
  - Apply your knowledge of a general category to more specific objects

Diagramming Inheritance Using the UML (1 of 4)

- **Unified Modeling Language (UML)**
  - Consists of many types of diagrams
- **Class diagram**
  - A visual tool
  - Provides an overview of a class

Diagramming Inheritance Using the UML (2 of 4)

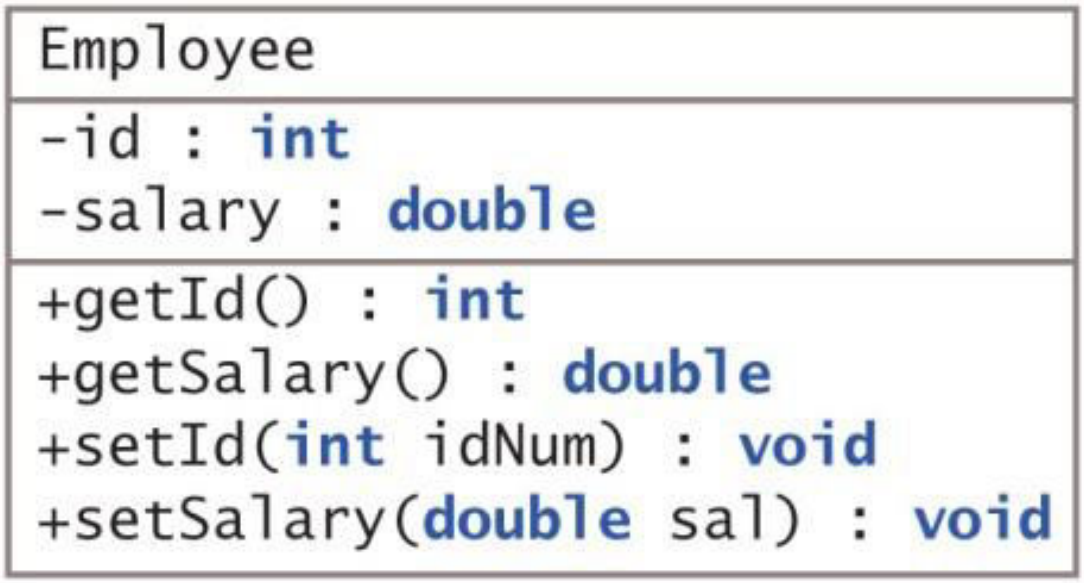


Figure 10-2 The Employee class diagram

Diagramming Inheritance Using the UML (3 of 4)

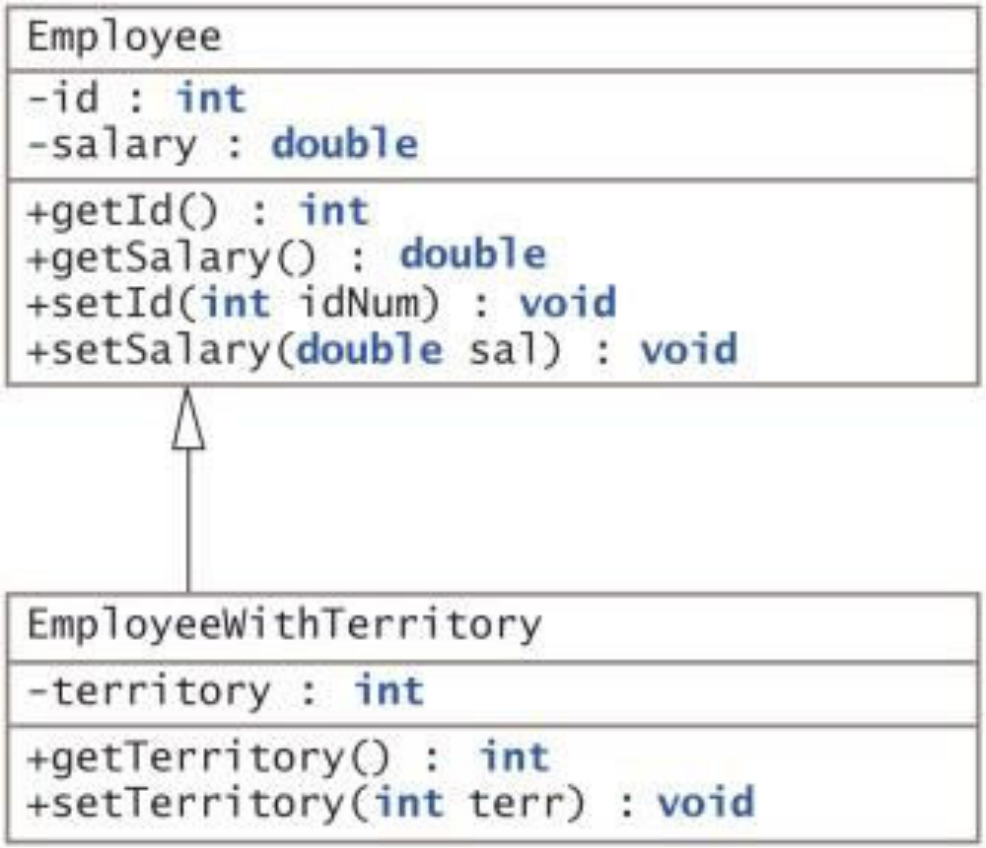


Figure 10-3 Class diagram showing the relationship between `Employee` and `EmployeeWithTerritory`

Diagramming Inheritance Using the UML (4 of 4)

- Use inheritance to create a derived class
  - Saves time
  - Reduces errors
  - Reduces the amount of new learning required to use a new class

Inheritance Terminology

- **Base class**
  - Used as a basis for inheritance
  - Also called:
    - **Superclass**
    - **Parent class**
- **Derived class**
  - Inherits from a base class
  - Always “is a” case or an example of a more general base class
  - Also called:
    - **Subclass**
    - **Child class**
- **Containment:** “has-a” inheritance
  - **Composition**
  - **Aggregation**

Extending Classes (1 of 2)

- Keyword **`extends`**
  - Used to achieve inheritance in Java
  - Example:

```
public class EmployeeWithTerritory extends Employee
```

- Inheritance is a one-way proposition
  - A child inherits from a parent, not the other way around
- Subclasses are more specific
- **`instanceof` operator**

Extending Classes (2 of 2)

```
public class EmployeeWithTerritory extends Employee
{
   private int territory;
   public int getTerritory()
   {
      return territory;
   }
   public void setTerritory(int terr)
   {
      territory = terr;
   }
}
```

Figure 10-4 The EmployeeWithTerritory class

Overriding Superclass Methods (1 of 2)

- Create a subclass by extending an existing class
  - A subclass contains data and methods defined in the original superclass
  - Sometimes superclass data fields and methods are not entirely appropriate for subclass objects
- **Polymorphism**
  - The same method name is used to indicate different implementations

Overriding Superclass Methods (2 of 2)

- **Override the method** in the parent class
  - Create a method in a child class that has the same name and parameter list as a method in its parent class
- **Subtype polymorphism**
  - The ability of one method name to work appropriately for different subclass objects of the same parent class
- **Override annotation (@Override)**
  - Tells compiler you are overriding a parent class method

Calling Constructors During Inheritance (1 of 3)

- When you instantiate an object that is a member of a subclass, you call at least two constructors:
  - The constructor for the base class
  - The constructor for the extended class
- The superclass constructor must execute first
- When the superclass contains a default constructor, the execution of the superclass constructor is transparent

Calling Constructors During Inheritance (2 of 3)

```
public class BaseballPlayer
{
   private int jerseyNumber;
   private double battingAvg;
   public static void showOrigins()
   {
      System.out.println("Abner Doubleday is often " +
         "credited with inventing baseball");
   }
}
```

Figure 10-18 The BaseballPlayer class

Calling Constructors During Inheritance (3 of 3)

```
In superclass constructor
In subclass constructor
```

Figure 10-9 Output of the `DemoConstructors` application

Using Superclass Constructors That Require Arguments (1 of 2)

- When you write your own constructor
  - You replace the automatically supplied version
- When extending a superclass with constructors that require arguments
  - The subclass must provide the superclass constructor with the arguments it needs

Using Superclass Constructors That Require Arguments (2 of 2)

- When a superclass has a default constructor
  - You can create a subclass with or without its own constructor
- When a superclass contains only constructors that require arguments
  - You must include at least one constructor for each subclass you create
    - The first statement within each constructor must call one of the superclass constructors
- Call the superclass constructor
  - `super(list of arguments);`
- Keyword **`super`**
  - Always refers to the superclass

Accessing Superclass Methods (1 of 2)

- Use the overridden superclass method within a subclass
  - Use the keyword `super` to access the parent class method

Accessing Superclass Methods (2 of 2)

```
public class PreferredCustomer extends Customer
{
   double discountRate;
   public PreferredCustomer(int id, double bal, double rate)
   {
      super(id, bal);
      discountRate = rate;
   }
   @Override
   public void display()
   {
      super.display();
      System.out.println(" Discount rate is " + discountRate);
   }
}
```

This statement calls the superclass display() method.

Figure 10-13 The PreferredCustomer class

Comparing `this` and `super`

- Think of the keyword `this` as the opposite of `super` within a subclass
- When a parent class contains a method that is not overridden
  - The child can use the method name with `super` or `this`, or alone

Employing Information Hiding (1 of 3)

- Within the `Student` class:
  - The keyword `private` precedes each data field
  - The keyword `public` precedes each method
- **Information hiding**
  - The concept of keeping data private
  - Data can be altered only by methods you choose and only in ways that you can control

Employing Information Hiding (2 of 3)

```
public class Student
{
   private int idNum;
   private double gpa;
   public int getIdNum()
   {
      return idNum;
   }
   public double getGpa()
   {
      return gpa;
   }
   public void setIdNum(int num)
   {
      idNum = num;
   }
   public void setGpa(double gradePoint)
   {
      gpa = gradePoint;
   }
}
```

Figure 10-17 The Student class

Employing Information Hiding (3 of 3)

- When a class serves as a superclass
  - Subclasses inherit all data and methods of the superclass
    - Except `private` members of the parent class are not accessible within a child class's methods
- Keyword **`protected`**
  - Provides an intermediate level of security between `public` and `private` access
  - Can be used within its own class or in any classes extended from that class
  - Cannot be used by "outside" classes

Methods You Cannot Override

- `static` methods
- `final` methods
- Methods within `final` classes

CENGAGE

A Subclass Cannot Override `static` Methods in Its Superclass (1 of 2)

- A subclass cannot override methods declared `static` in the superclass
- A subclass can hide a `static` method in the superclass by declaring a `static` method with the same signature as the `static` method in the superclass
  - Then call the new `static` method from within the subclass or in another class by using a subclass object
  - Within the `static` method of a subclass, you cannot access the parent method using the `super` object
- Although a child class cannot inherit its parent's `static` methods, it can access its parent's `static` methods in the same way any other class can

A Subclass Cannot Override `static` Methods in Its Superclass (2 of 2)

```
public class ProfessionalBaseballPlayer extends BaseballPlayer
{
   double salary;
   @Override
   public void showOrigins()
   {
      super.showOrigins();
      System.out.println("The first professional " +
          "major league baseball game was played in 1871");
   }
}
```

Don't Do It
A nonstatic method cannot override a `static` member of a parent class.

Figure 10-19 The `ProfessionalBaseballPlayer` class attempting to override the parent's `static` method

A Subclass Cannot Override `final` Methods in Its Superclass (1 of 2)

- A subclass cannot override methods declared `final` in the superclass
- `final` modifier
  - Does not allow the method to be overridden
- **Virtual method calls**
  - Default in Java
  - The method used is determined when the program runs
  - The object type might not be known until the method executes

A Subclass Cannot Override `final` Methods in Its Superclass (2 of 2)

- Advantages to making the method `final`:
  - The compiler knows only one version of the method exists
  - The compiler knows which method version will be used
  - A program's performance can be optimized by removing calls to `final` methods
    - **Inlining** the code: each method call is replaced with the expanded code of the method's definition

A Subclass Cannot Override Methods in a `final` Superclass (1 of 2)

- When a class is declared `final`:
  - All of its methods are `final` regardless of which access modifier precedes the method name
  - It cannot be a parent class

A Subclass Cannot Override Methods in a `final` Superclass (2 of 2)

```
public final class HideAndGoSeekPlayer
{
   private int count;
   public void displayRules()
   {
      System.out.println("You have to count to " + count +
         " before you start looking for hiders");
   }
}
public final class ProfessionalHideAndGoSeekPlayer
   extends HideAndGoSeekPlayer
{
   private double salary;
}
```

Notice the keyword `final` in the method header.

Don't Do It
You cannot extend a `final` class.

Figure 10-29 The `HideAndGoSeekPlayer` and `ProfessionalHideAndGoSeekPlayer` classes

Don’t Do It

- Don’t capitalize the *o* in the `instanceof` operator
- Don’t try to directly access private superclass members from a subclass
- Don’t forget to call a superclass constructor from within a subclass constructor if the superclass does not contain a default constructor
- Don’t try to override a `final` method in an extended class
- Don’t try to extend a `final` class

Summary (1 of 2)

- Inheritance
  - A mechanism that enables one class to inherit both the behavior and the attributes of another class
- Keyword `extends`
  - Used to achieve inheritance in Java
- Polymorphism
  - The act of using the same method name to indicate different implementations

Summary (2 of 2)

- Use a superclass method within a subclass
  - Use the keyword `super` to access it
- Information hiding
  - The concept of keeping data private
- Keyword `protected`
  - Provides an intermediate level of security between `public` and `private` access
- A subclass cannot override methods that are:
  - Declared `static` in a superclass
  - Declared `final` or declared within a `final` class