

JAVATM PROGRAMMING



Chapter 10: Introduction to Inheritance



Objectives

- Learn about 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

- **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 (cont'd.)

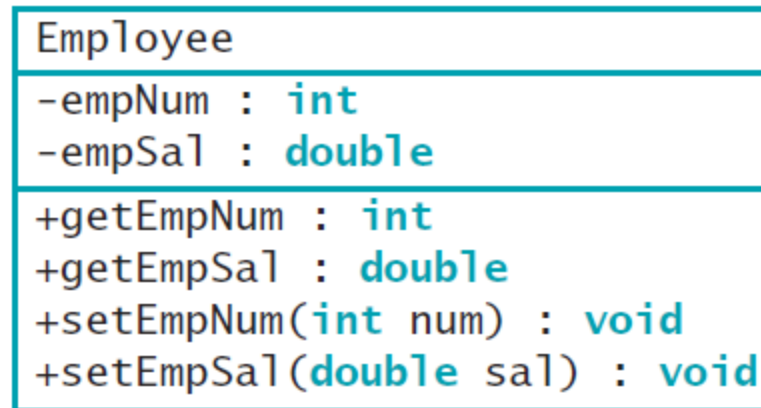


Figure 10-2 The Employee class diagram

Diagramming Inheritance Using the UML (cont'd.)

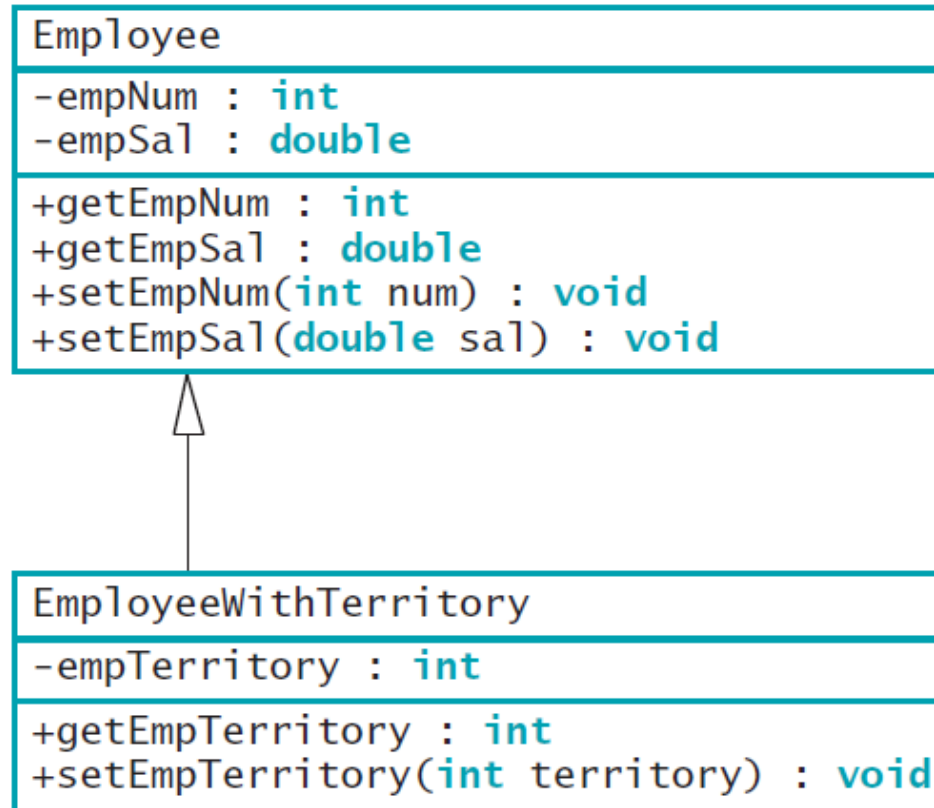


Figure 10-3 Class diagram showing the relationship between **Employee** and **EmployeeWithTerritory**

Diagramming Inheritance Using the UML (cont'd.)

- Use inheritance to create a derived class
 - Save time
 - Reduce errors
 - Reduce 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**



Inheritance Terminology (cont'd.)

- **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**



Extending Classes

- 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 (cont'd.)

```
public class EmployeeWithTerritory extends Employee
{
    private int empTerritory;
    public int getEmpTerritory()
    {
        return empTerritory;
    }
    public void setEmpTerritory(int num)
    {
        empTerritory = num;
    }
}
```

Figure 10-4 The EmployeeWithTerritory class



Overriding Superclass Methods

- 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**
 - Using the same method name to indicate different implementations



Overriding Superclass Methods (cont'd.)

- **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



Calling Constructors During Inheritance

- 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 (cont'd.)

```
public class ASuperClass
{
    public ASuperClass()
    {
        System.out.println("In superclass constructor");
    }
}
public class ASubClass extends ASuperClass
{
    public ASubClass()
    {
        System.out.println("In subclass constructor");
    }
}
public class DemoConstructors
{
    public static void main(String[] args)
    {
        ASubClass child = new ASubClass();
    }
}
```

Figure 10-8 Three classes that demonstrate constructor calling when a subclass object is instantiated

Calling Constructors During Inheritance (cont'd.)



```
C:\Java>java DemoConstructors
In superclass constructor
In subclass constructor
C:\Java>
```

Figure 10-9 Output of the `DemoConstructors` application

Using Superclass Constructors That Require Arguments

- 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 (cont'd.)

- 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

Using Superclass Constructors That Require Arguments (cont'd.)

- Call the superclass constructor
 - `super(list of arguments);`
- Keyword **super**
 - Always refers to the superclass



Accessing Superclass Methods

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

Accessing Superclass Methods (cont'd.)

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

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

- 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 (cont'd.)

```
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 10-16 The Student class



Employing Information Hiding (cont'd.)

- 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

Employing Information Hiding (cont'd.)

- 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

A Subclass Cannot Override `static` Methods in Its Superclass

- 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

A Subclass Cannot Override `static` Methods in Its Superclass (cont'd.)

- 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 (cont'd.)

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

Figure 10-22 The ProfessionalBaseballPlayer class

A Subclass Cannot Override `final` Methods in Its Superclass

- 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 type of object used might not be known until the method executes

A Subclass Cannot Override `final` Methods in Its Superclass (cont'd.)

- Advantages to making the method `final`:
 - The compiler knows there is only one version of the method
 - The compiler knows which method version will be used
 - It can optimize a program's performance by removing calls to `final` methods
 - Replaces them with expanded code of their definitions at each method call location
 - Called **inlining** the code

A Subclass Cannot Override Methods in a `final` Superclass

- 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 (cont'd.)

```
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;
}
```

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

Figure 10-28 The `HideAndGoSeekPlayer` and `ProfessionalHideAndGoSeekPlayer` classes



You Do It

- Demonstrating Inheritance
- Overriding a Superclass Method
- Understanding the Role of Constructors in Inheritance



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

- 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 (cont'd.)

- 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