NCP2103: Object-Oriented Programming (Java Programming)

Introduction to Inheritance

Module 9

Errol John M. Antonio
Assistant Professor
Computer Engineering Department
University of the East – Manila Campus

Objectives

- Learn about the concept of inheritance
- Extend classes
- Override superclass methods
- Call constructors during inheritance

Objectives (cont'd.)

- Access superclass methods
- Employ information hiding
- Learn which methods you cannot override

Learning About the Concept of Inheritance

- Inheritance
  - Mechanism that enables one class to inherit behavior and attributes of another class
  - Apply knowledge of general category to more specific objects

Learning About the Concept of Inheritance (cont'd.)

- Unified Modeling Language (UML)
  - Consists of many types of diagrams
- Class diagram
  - Visual tool
  - Provides overview of a class

Learning About the Concept of Inheritance (cont'd.)

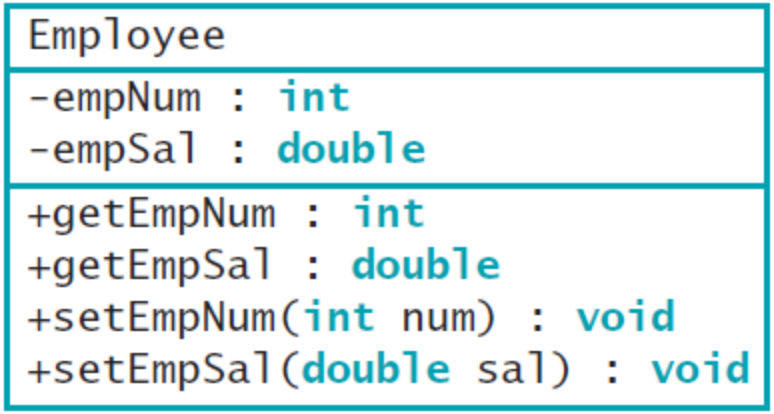

Figure 10-2 The Employee class diagram

Learning About the Concept of Inheritance (cont'd.)

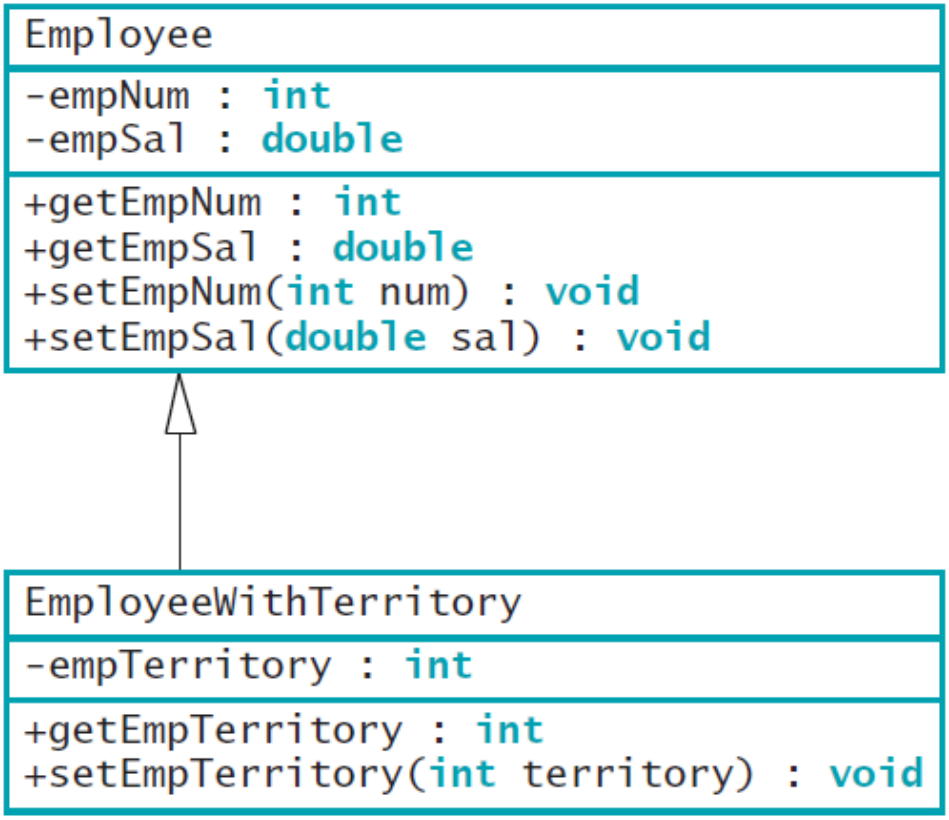

Figure 10-3 Class diagram showing the relationship between Employee and EmployeeWithTerritory

Learning About the Concept of Inheritance (cont'd.)

- Use inheritance to create derived class
  - Save time
  - Reduce errors
  - Reduce amount of new learning required to use 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 example of more general base class
  - Also called:
    - Subclass
    - Child class

Extending Classes

- Keyword `extends`
  - Achieve inheritance in Java
  - Example:

```
public class EmployeeWithTerritory
extends Employee
```

- Inheritance is a one-way proposition
  - Child inherits from parent, not other way around
- Subclasses are more specific
- `instanceof` keyword

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 subclass by extending existing class
  - Subclass contains data and methods defined in original superclass
  - Sometimes superclass data fields and methods are not entirely appropriate for subclass objects
- Polymorphism
  - Using same method name to indicate different implementations

Overriding Superclass Methods (cont'd.)

- Override method in parent class
  - Create method in child class that has same name and parameter list as method in parent class
- Subtype polymorphism
  - Ability of one method name to work appropriately for different subclass objects of same parent class

Call Constructors During Inheritance

- Instantiate object that is member of subclass
  - Call at least two constructors
    - Constructor for base class
    - Constructor for extended class
  - Superclass constructor must execute first
- When superclass contains default constructor
  - Execution of superclass constructor transparent

```
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 10-5 Three classes that demonstrate constructor calling when a subclass object is instantiated

Call Constructors During Inheritance (cont'd.)

Figure 10-6 Output of the `DemoConstructors` application

Using Superclass Constructors that Require Arguments

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

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

- When superclass has default constructor
  - Can create subclass with or without own constructor
- When superclass contains only constructors that require arguments
  - Must include at least one constructor for each subclass you create
  - First statement within each constructor must call superclass constructor

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

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

Accessing Superclass Methods

- Use overridden superclass method within subclass
  - Use keyword `super` to access parent class method
- Comparing `this` and `super`
  - Think of the keyword `this` as the opposite of `super`
    - Within a subclass
  - When parent class contains a method that is not overridden
    - Child can use the method name with `super`, `this`, or alone

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 10-8 The PreferredCustomer class

Comparing `this` and `super`

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

Employing Information Hiding

- Student class
  - Keyword `private` precedes each data field
  - Keyword `public` precedes each method
- Information hiding
  - Concept of keeping data private
  - Data can be altered only by methods you choose and only in ways that you can control

```
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 10-11 The Student class

Employing Information Hiding (cont'd.)

- When class serves as superclass
  - Subclasses inherit all data and methods of superclass
  - Except `private` members of parent class not accessible within child class's methods

Employing Information Hiding (cont'd.)

- Keyword `protected`
  - Provides intermediate level of security between `public` and `private` access
  - Can be used within 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

- Subclass cannot override methods declared `static` in superclass
- Can hide `static` method in superclass
  - By declaring `static` method with same signature as `static` method in superclass
  - Call new `static` method from within subclass or in another class by using subclass object
  - Within `static` method of subclass
    - Cannot access parent method using `super` object

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

- Although child class cannot inherit parent's static methods
  - Can access 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 10-17 The ProfessionalBaseballPlayer class

A Subclass Cannot Override `final` Methods in Its Superclass

- Subclass cannot override methods declared `final` in superclass
- `final` modifier
  - Does not allow method to be overridden
- Virtual method calls
  - Default in Java
  - Method used is determined when program runs
  - Type of object used might not be known until method executes

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

- Advantage to making method `final`
  - Compiler knows there is only one version of method
  - Compiler knows which method version will be used
  - Can optimize program's performance
    - By removing calls to final methods
    - Replacing them with expanded code of their definitions
    - At each method call location
    - Called inlining

A Subclass Cannot Override Methods in a `final` Superclass

- Declare class `final`
  - All of its methods are `final`
  - Regardless of which access modifier precedes method name
  - 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 10-23 The `HideAndGoSeekPlayer` and `ProfessionalHideAndGoSeekPlayer` classes

You Do It

- Creating a superclass and an application to use it
- Creating a subclass and an application to use it
- Understanding the role of constructors in inheritance
- Inheritance when the superclass requires constructor arguments

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

Summary

- Inheritance
  - Mechanism that enables one class to inherit both behavior and attributes of another class
- Keyword `extends`
  - Achieve inheritance in Java
- Polymorphism
  - Act of using same method name to indicate different implementations

Summary (cont'd.)

- Use a superclass method within a subclass
  - Use keyword `super` to access it
- Information hiding
  - Concept of keeping data private
- Keyword `protected`
  - Intermediate level of security between `public` and `private` access
- Subclass cannot override methods
  - Declared `static` in superclass
  - Declared `final` or class `final`

End of Module.

REFERENCE:

Farrell, J. (2016). *Java Programming*. 8th Edition. Course Technology, Cengage Learning.