Lecture 06 - Inheritance

CS202: Introduction to Object Oriented Programming
Victor Mejia
CSULA

Today's Topics:

- Introduction - Inheritance
- Superclasses and Subclasses
- Using the `super` Keyword
- Overriding Methods
- The `Object` Class
- Polymorphism and Dynamic Binding
- Casting objects and the instanceof

Introduction

Motivations

Suppose you work for a company at which managers are treated differently from other employees:

- both employees are managers are paid a salary
- managers get bonuses
- a manager is also an employee

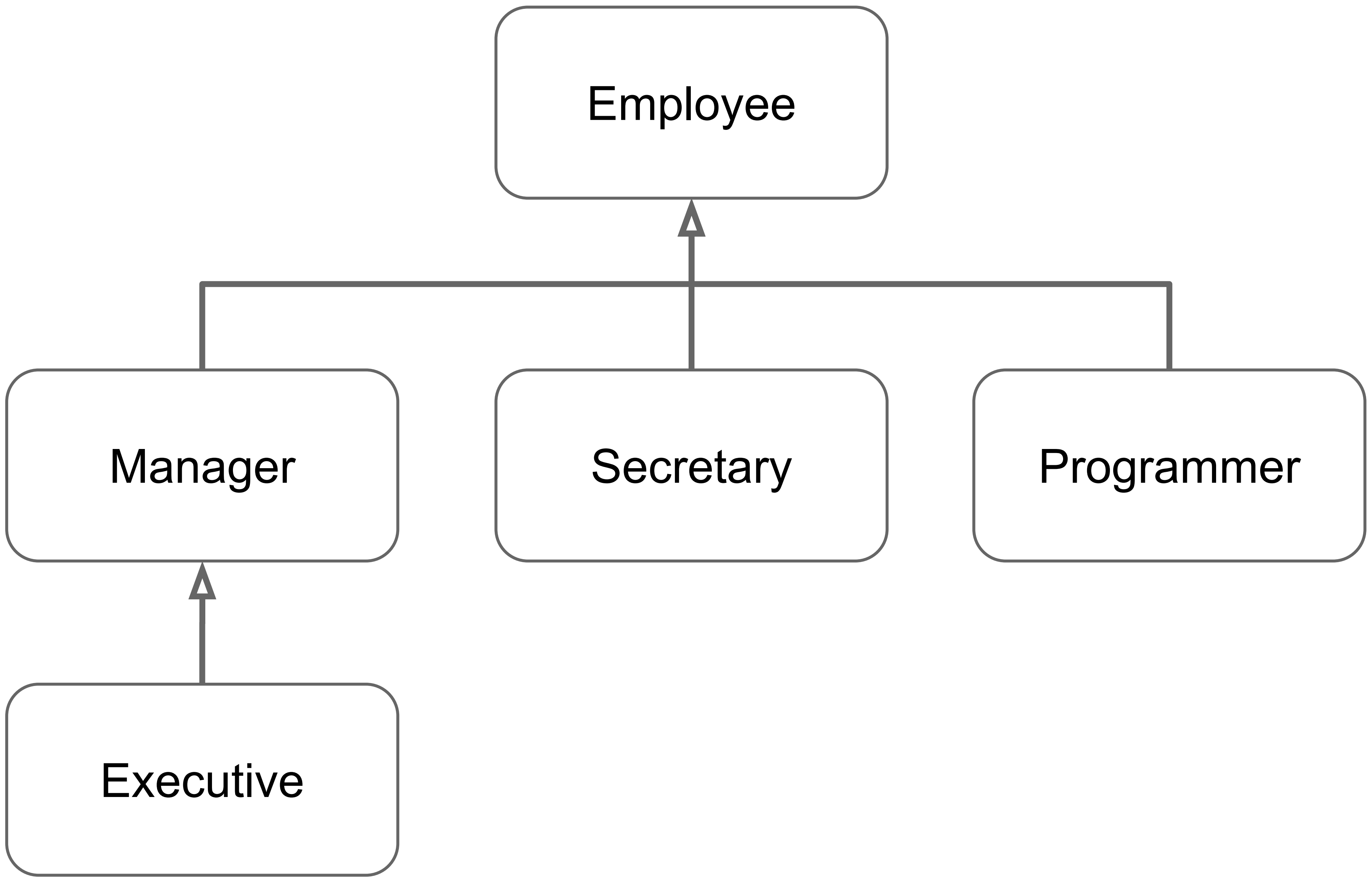
How would we model this?

Inheritance

Inheritance enables you to

- define a general class (i.e., a superclass)
- later extend it to more specialized classes (i.e., subclasses).
- Inheritance models the “is-a” relationship
- Every manager “is an” employee.

Inheritance - UML Diagram



Inheritance

subclass   superclass

```
public class Manager extends Employee {
   added methods and fields
}
```

- **extends**: you are making a new class that derives from an existing class
- existing class: **superclass**, base class, parent class
- derived class: **subclass**, derived class, child class

Inheritance

subclass → Manager

superclass → Employee

```
public class Manager extends Employee {
   added methods and fields
}
```

- The Employee class is a superclass, but not because it is superior to its subclass or contains more functionality.
- The opposite is true: subclasses have more functionality than their superclasses.

Superclasses and Subclasses

subclass superclass

```
public class Manager extends Employee {
   added methods and fields
}
```

- The keyword extends tells the compiler that the `Manager` class extends the `Employee` class, thus inheriting the methods `getName, getHireDay.`
- `even though these methods are not explicitly defined in the Manager class, they are automatically inherited (so are the fields).`

Defining a Subclass

A subclass inherits from a superclass. You can also:

- Add new properties
- Add new methods
- Override the methods of the superclass

Inheritance Example

Employee.java

Manager.java

Using the super Keyword

Using the Keyword `super`

The keyword `super` refers to the superclass of the class in which `super` appears. This keyword can be used in two ways:

- To call a superclass constructor
- To call a superclass method

Using the Keyword super

The Manager’s getSalary() method should add the bonus.

```
public double getSalary() {
   return this.salary + this.bonus; // won’t work
}
```

won’t work because the Manager subclass has no direct access to the private fields of the superclass.

Using the Keyword **super**

The Manager’s getSalary() method should add the bonus.

```
public double getSalary() {
   return getSalary() + this.bonus; // won’t work
}
```

won’t work because getSalary() would be calling itself, resulting in a stack overflow

Using the Keyword super

The Manager’s getSalary() method should add the bonus.

```
public double getSalary() {
   return super.getSalary() + this.bonus;
}
```

works, calling the method on the superclass

Using the Keyword `super`

The keyword super is not a reference to an object. For example you cannot do this:

```
super.salary = x;
```

It is a special keyword that direct the compiler to invoke the superclass method.

Are constructors of the superclass inherited?

No. They are not inherited.

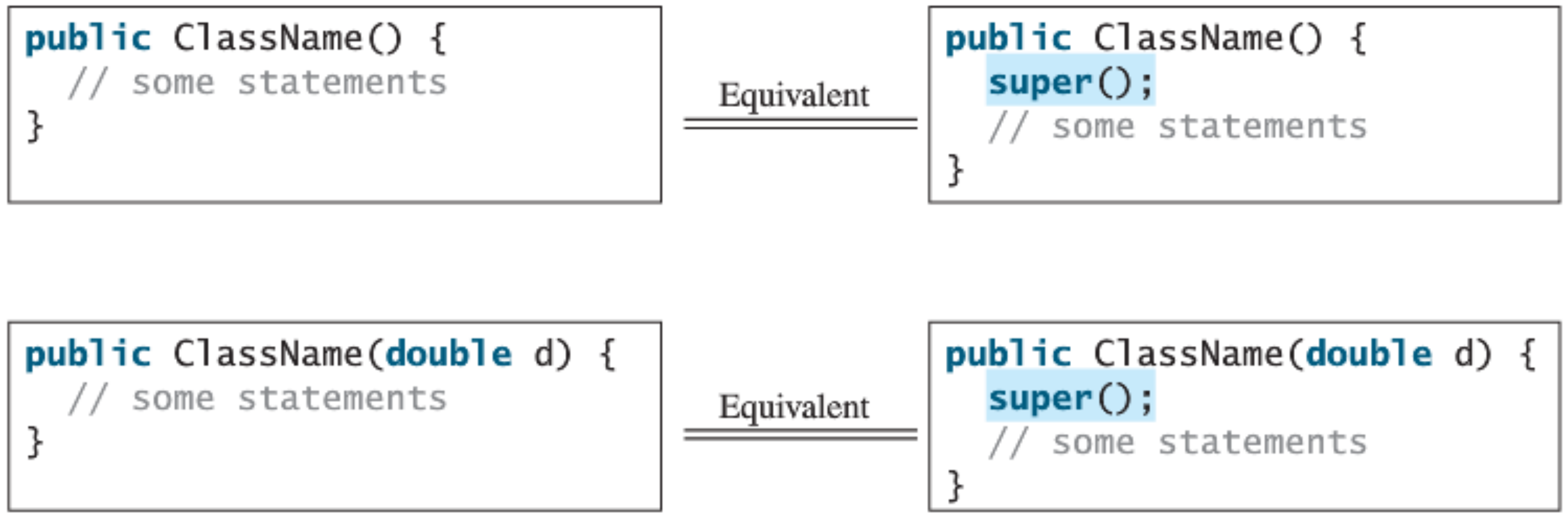
They are invoked explicitly or implicitly.

Explicitly using the super keyword.

A constructor is used to construct an instance of a class. Unlike properties and methods, a superclass's constructors are not inherited in the subclass. They can only be invoked from the subclasses' constructors, using the keyword <u>super</u>. *If the keyword <u>super</u> is not explicitly used, the superclass's no-arg constructor is automatically invoked.*

Superclass's Constructor Is Always Invoked

A constructor may invoke an overloaded constructor or its superclass's constructor. If none of them is invoked explicitly, the compiler puts super() as the first statement in the constructor. For example,



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

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

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

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

CAUTION

You must use the keyword super to call the superclass constructor. Invoking a superclass constructor’s name in a subclass causes a syntax error. Java requires that the statement that uses the keyword super appear first in the constructor.

Constructor Chaining

Constructing an instance of a class invokes all the superclasses’ constructors along the inheritance chain. This is known as *constructor chaining*.

```
public class Employee extends Person {
  public Employee() {
    this("(2) Invoke Employee’s overloaded constructor");
    System.out.println("(3) Employee's no-arg constructor is invoked");
  }

  public Employee(String s) {
    System.out.println(s);
  }
}

public class Employee {
  public Person() {
    System.out.println("(1) Person's no-arg constructor is invoked");
  }
}
```

Example on the Impact of a Superclass without no-arg Constructor

Find out the errors in the program:

```
public class Apple extends Fruit {
}

class Fruit {
  public Fruit(String name) {
    System.out.println("Fruit's constructor is invoked");
  }
}
```

Example on the Impact of a Superclass without no-arg Constructor

Find out the errors in the program:

```
public class Apple extends Fruit {
}

class Fruit {
  public Fruit(String name) {
    System.out.println("Fruit's constructor is invoked");
  }
}
```

Since no constructor is explicitly defined in `Apple`, `Apple`'s default no-arg constructor is defined implicitly. Since `Apple` is a subclass of `Fruit`, `Apple`'s default constructor automatically invokes `Fruit`'s no-arg constructor. However, `Fruit` does not have a no-arg constructor, because `Fruit` has an explicit constructor defined. Therefore, the program cannot be compiled.

Defining a Subclass

A subclass inherits from a superclass. You can also:

- Add new properties
- Add new methods
- Override the methods of the superclass

Overriding Methods

Overriding Methods in the Superclass

A subclass inherits methods from a superclass. Sometimes it is necessary for the subclass to modify the implementation of a method defined in the superclass. This is referred to as *method overriding*.

```
public class Circle extends GeometricObject {

  // Other methods are omitted


  /** Override the toString method defined in GeometricObject */
  public String toString() {
    return super.toString() + "\nradius is " + radius;
  }

}
```

NOTE

- An instance method can be overridden only if it is accessible.
- Thus a private method cannot be overridden, because it is not accessible outside its own class.
- If a method defined in a subclass is private in its superclass, the two methods are completely unrelated.

NOTE

- Like an instance method, a static method can be inherited.
- However, a static method cannot be overridden.
- If a static method defined in the superclass is redefined in a subclass, the method defined in the superclass is hidden.

Overriding vs. Overloading

```
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);
  }
}
```

```
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);
  }
}
```

Polymorphism

Polymorphism

Simple rule for inheritance:

- The “is-a” rule states that every object of the subclass is an object of the superclass
- Every manager is an employee
- Manager is a subclass of the Employee class (the opposite is not true)
- substitution principle: you can use a subclass object whenever the program expects a superclass object

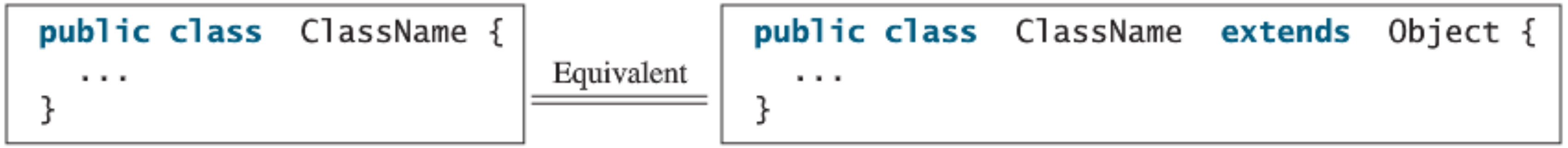
Polymorphism

```
Employee e;
e = new Employee(...); // Employee object expected
e = new Manager(...); // OK, Manager can be used as well
```

polymorphism: an object variable can refer to multiple actual types

The <u>Object</u> Class and Its Methods

Every class in Java is descended from the java.lang.Object class. If no inheritance is specified when a class is defined, the superclass of the class is Object.



```
public class  ClassName {
  ...
}
```

```
public class  ClassName  extends  Object {
  ...
}
```

The toString() method in Object

The toString() method returns a string representation of the object. The default implementation returns a string consisting of a class name of which the object is an instance, the at sign (@), and a number representing this object.

```
Loan loan = new Loan();
System.out.println(loan.toString());
```

The code displays something like `Loan@15037e5` . This message is not very helpful or informative. Usually you should override the toString method so that it returns a digestible string representation of the object.