Inheritance

Part 1 – Inheritance Basics

Chapter 5, Core Java Volume I and

Chapter 10, Java How to Program, 10th ed.

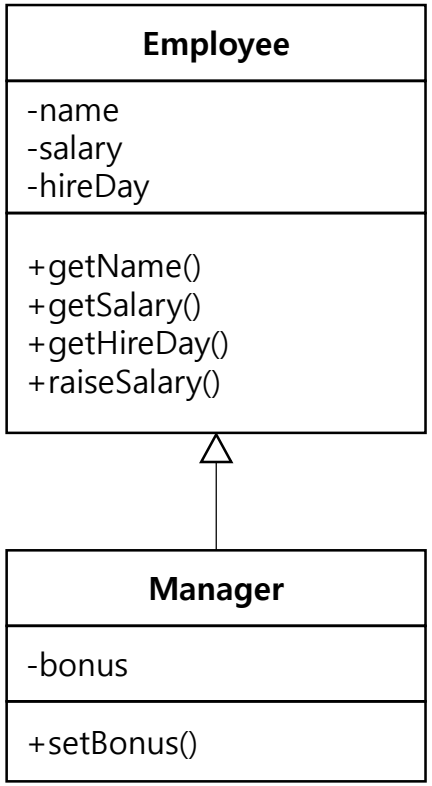
Contents

- Superclasses and Subclasses
- Defining Subclasses
- Subclass Constructors
- Using Subclasses
- Overriding Methods
- Exmaple : Employee and Manager
- More Examples
- Multiple Inheritance

Superclasses and Subclasses

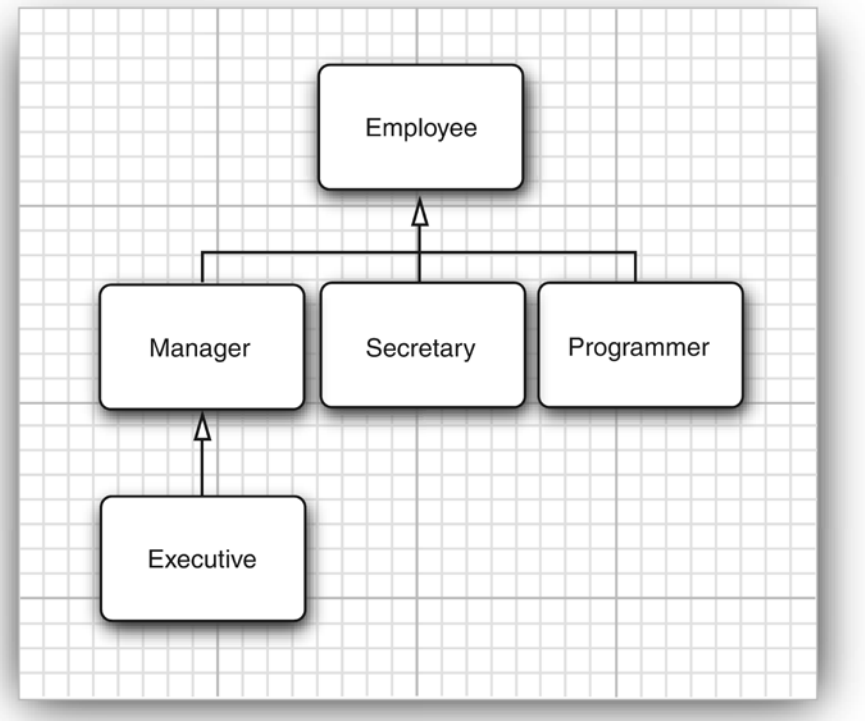
- We can create a *new class* from an *existing class*.
- The new class *inherits* features (instance variables and methods) from an existing class.

- Example: Managers are in many aspects like employees.
  - However, in other aspects they are different:
  - Managers gets a bonus.
- Every manager is an employee, but not every employee is a manager.
- The class of managers is a *subclass* of an existing employee class.
- The Employee class is a *superclass*.



Inheritance Hierarchies

- Superclass (base class, parent class)
  - direct superclass
  - indirect superclass
- Subclass (derived class, child class, exended class)
  - direct subclass
  - indirect subclass



Example : an Employee class (revisit)

```
class Employee
{
   // Fields
   private String name;
   private double salary;
   private LocalDate hireDay;

   // Constructors
   public Employee(String n, double s, int year, int month,
      int day)
   {
      name = n;
      salary = s;
      hireDay = LocalDate.of(year, month, day);
   }
```

```
   // Methods
   public String getName()
   {
      return name;
   }
   public double getSalary()
   {
      return salary;
   }
   public LocalDate getHireDay()
   {
      return hireDay;
   }
   public void raiseSalary(double byPercent)
   {
      double raise = salary * byPercent / 100;
      salary += raise;
   }
}
```

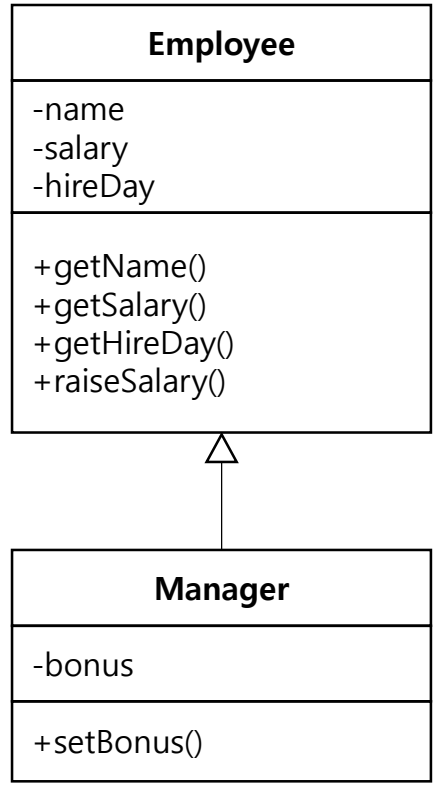
Defining Subclasses

- Step 1: Use the extends keyword to define subclasses

```
public class Manager extends Employee
{
 // added methods and fields unique to managers class
 }
```

- Step 2: Add fields and methods:

```
public class Manager extends Employee
{
    private double bonus;                    // added fields
   . . . ……..;
    public void setBonus(double bonus)  // added methods
    {
      this.bonus = bonus;
    }
 }
```



Inheritance Hierarchy

Defining Subclasses

- Step3: Define Constructors
  - Subclass constructor can invoke superclass constructor:

```
public Manager(String name, double salary, int year, int month, int day)
{
   super(name, salary, year, month, day);
   bonus = 0;
}
```

  - Call using super must be the first statement.
- If no explicit call to superclass constructor, no-arg constructor of superclass is invoked.
  - If the superclass does not have a no-arg constructor, the compiler reports an error.

Defining Subclasses

- Caution: Subclasses cannot access the private data or methods in superclasses

```
public String toString()
{
    return "Manager: " + name; ; // won't work
}
```

- Subclasses have to call public methods of the superclass to access the private data in superclasses

```
public String toString()
{
    return "Manager: " + getName() ;
}
```

Using Subclasses

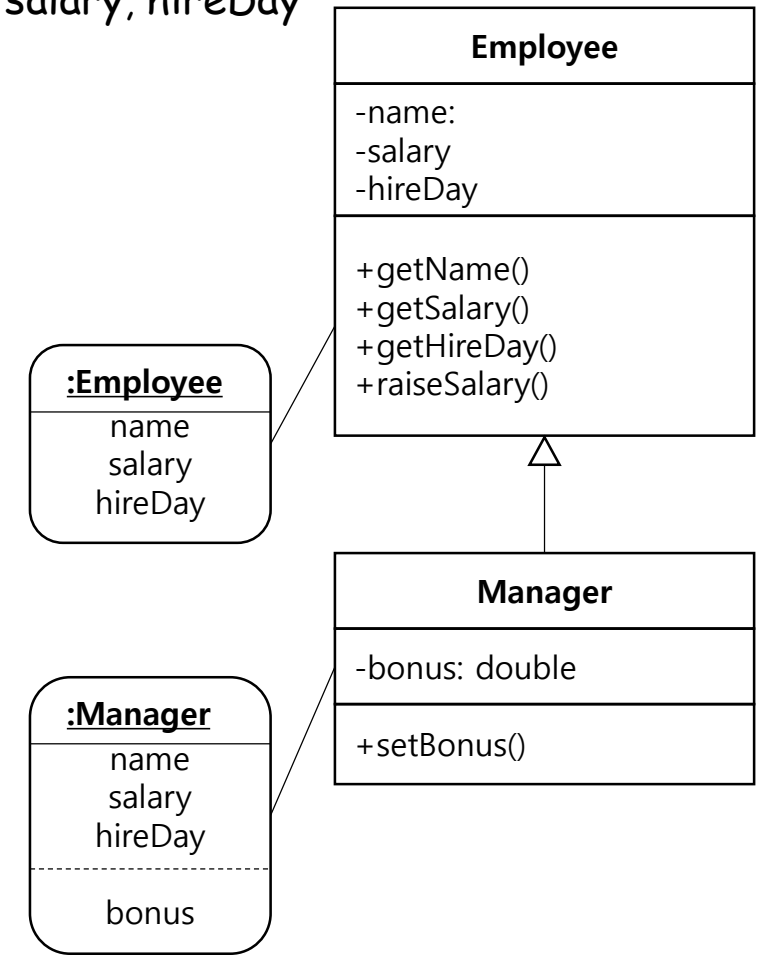
- Subclass inherits methods and instance variables from superclass;
  - Class Manager inherits getName, getHireday, getSalary, and raiseSalary from class Employee
  - Class Manager inherits fields from Employee superclass: name, salary, hireDay

- You can invoke all the public methods of the superclass on the object of its subclass.

```
Manager boss = new Manager("Carl Cracker", 80000, 1987, 12, 15);
boss.setBonus(5000);    // calling a method in Manager
boss.raiseSalary(10);   // calling a method in Employee
```

- You cannot call a subclass's method to the object of its superclass.

```
Employee emp = new Employee("Carl Cracker", 80000, 1987, 12, 15);
emp.setBonus(5000);     // error
```



Overriding Methods

- When an inherited method is not appropriate, need to override it in the subclass.
- Example: overriding toString() in Manager

```
// in Employee

public String toString()
{
    return "Employee: " + name;
}
```

- First attempt: access to private fields

```
public String toString()
{
    return "Manager: " + name;   // won't work
}
```

- Second attempt: call public methods (getName() method in Employee)

```
public String toString()
{
    return "Manager: " + getName();
}
```

Overriding Methods

- When an overriding method calls the method to be overrided, you can use **super** keyword.
- Example: overriding getSalary()

  - First attempt: recursive call

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

  - Second attempt: using **super**

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

```
Manager boss =
    new Manager("Carl Cracker", 80000, 1987, 12, 15);
boss.getSalary(5000);   // calling a method in Manager
```

Example : Employee and Manager

```
package inheritance;
import java.time.*;

public class Employee
{
    private String name;
    private double salary;
    private LocalDate hireDay;
   public Employee(String name, double salary, int year,
     int month, int day)
    {
       this.name = name;
      this.salary = salary;
      hireDay = LocalDate.of(year, month, day);
    }// end of constuctor

}
```

```
public String getName()
 {
     return name;
 }
 public double getSalary()
 {
     return salary;
 }
 public LocalDate getHireDay()
 {
     return hireDay;
 }
 public void raiseSalary(double byPercent)
 {
    double raise = salary * byPercent / 100;
     salary += raise;
 }
```

Example : Employee and Manager

```
package inheritance;
public class Manager extends Employee
{
    private double bonus;
    public Manager(String name, double salary, int year, int month, int day)
    {
        super(name, salary, year, month, day);        // call super class's constructor
         bonus = 0;
     }
     public double getSalary() {                      // overriding
        double baseSalary = super.getSalary();
        return baseSalary + bonus;
    }
    public void setBonus(double b) {
        bonus = b;
    }
} // end of Manager
```

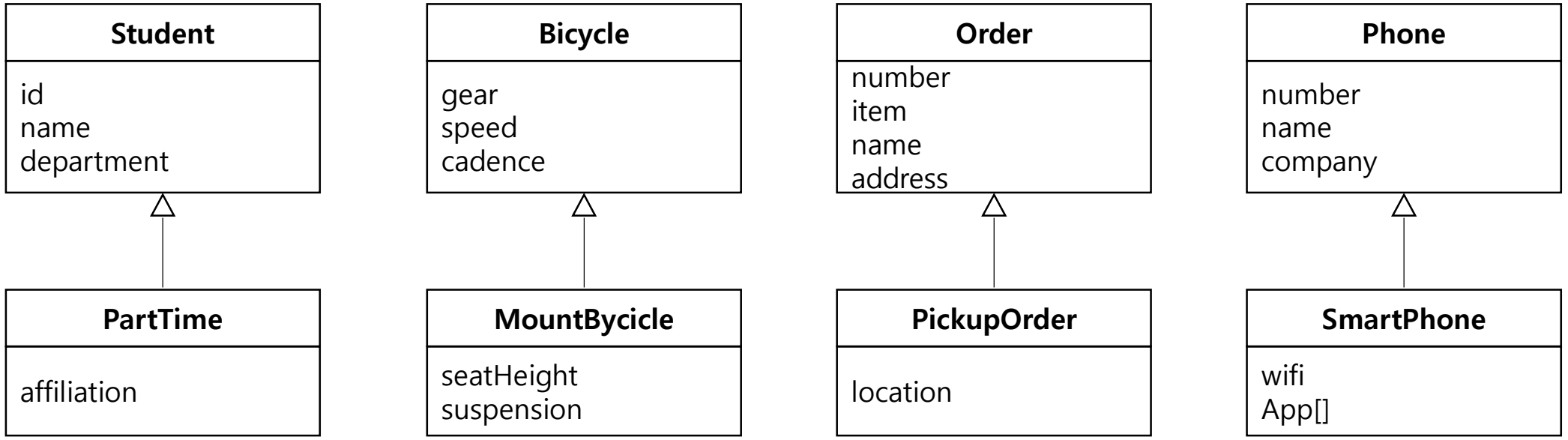
Example : Employee and Manager

```
package inheritance;
public class ManagerTest
{
   public static void main(String[] args)
   {
      Employee harry = new Employee("Harry Hacker", 50000, 1989, 10, 1);
      Employee tommy = new Employee("Tommy Tester", 40000, 1990, 3, 15);
      Manager boss = new Manager("Carl Cracker", 80000, 1987, 12, 15);
      boss.setBonus(5000);
      System.out.println("name=" + harry.getName() + ",salary=" + harry.getSalary());
      System.out.println("name=" + tommy.getName() + ",salary=" + tommy.getSalary());
      System.out.println("name=" + boss.getName() + ",salary=" + boss.getSalary());
   }
}
```

Compiler and Execution

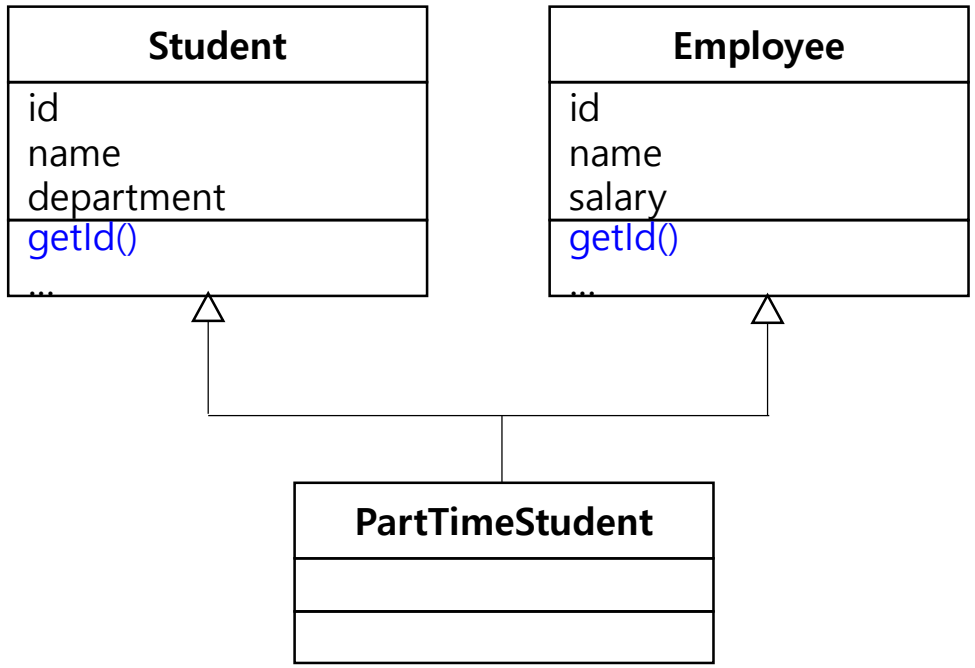
```
C:\corejava> javac -d . *.java
C:\corejava> java inheritance.ManagerTest
```

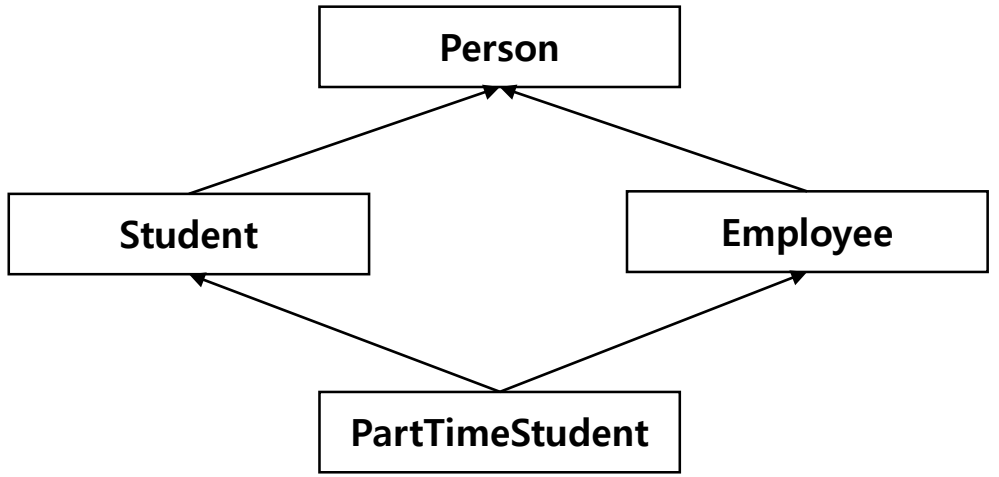
More Examples



Multiple Inheritance

- In single inheritance, a class has only one direct superclass, but multiple direct subclasses
- In multiple inheritance, a class can have more than one direct superclass.
- A subclass inherits all the properties from its superclasses.
- Java does not support multiple inheritance (cf. C++, Python, Perl, R, etc)
- Problems in multiple inheritance





Diamond Problem in Multiple Inheritance