Polymorphism and Interfaces

Week 04, Session 01
September 26, 2006

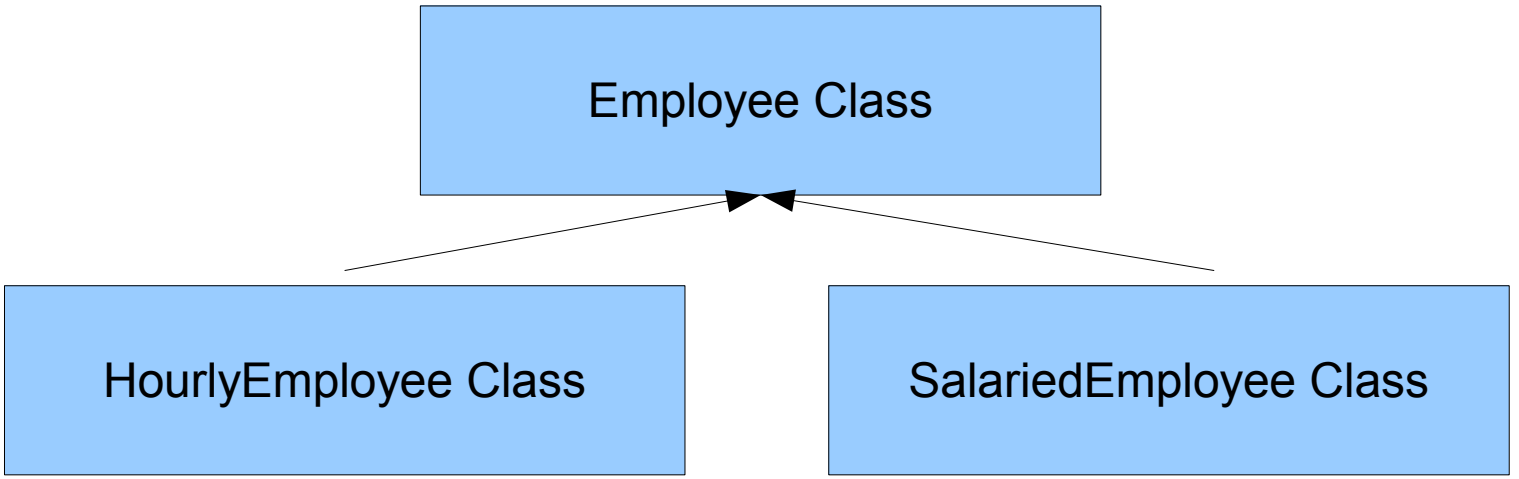
Politeknik Informatika Del
Jl. Sisingamangaraja, Sitoluama, Laguboti
Toba Samosir, Sumatera Utara, 22381
Indonesia

Gandhi M T Manalu
gandhi [at] del [dot] ac [dot] id
http://www.del.ac.id/~gandhi
IF21311 – Java

Overview

- The need of Polymorphism
- What's Polymorphism actually?
- Method Overloading
- Method Overriding thorugh Inheritance
- Runtime Polymorphism
- Overview of Interfaces
- Method Overriding thorugh Interface

The need of Polymorphism



Suppose we need a method to calculate salary of employee, where would you put it?
Of course, salary for Hourly Employee is different with salary for Salaried Employee, how to handle this?

What's Polymorphism actually?

- The meaning of the word polymorphism is something like one name, many forms
- Polymorphism manifests itself in Java in the form of multiple methods having the same name.
- In some cases, multiple methods have the same name, but different formal argument lists (overloaded methods).
- In other cases, multiple methods have the same name, same return type, and same formal argument list (overridden methods).

What's Polymorphism actually?

- From a practical programming viewpoint, polymorphism manifests itself in three distinct forms in Java:
  - Method overloading
  - Method overriding through inheritance
  - Method overriding through the Java interface

Method Overloading

- Method overloading:
  - A class may have two methods with the same name but have different type signatures (return type, arguments types and orders)
  - Subclass have a method with the same name with a method in its parent but have different type signatures

Method Overloading

```
public class MethodOverloadingSample {
    public void myMethod(int i) {
        System.out.println("Value passed to me: " + i);
    }

    public int myMethod(int i) {
        return (i + 1);
    }

    public static void main(String args[]) {
        MethodOverloadingSample mos = new MethodOverloadingSample();
        mos.myMethod(633);

        int k = mos.myMethod(k);
        System.out.println("Value returned from myMethod: " + k);
    }
}
```

Is the above code a valid Method Overloading?

Method Overloading

```
public class MethodOverloadingSample2 {
    public void show() {
        System.out.println("I'm [show] method without argument");
    }

    public void show(int i) {
        System.out.println("I'm [show] method with one integer argument: " + i);
    }

    public int show(String s, int i) {
        System.out.println(s);
        return (i + 5);
    }

    public static void main(String args[]) {
        MethodOverloadingSample mos = new MethodOverloadingSample();
        mos.show();
        mos.show(7);

        int retI = mos.show("Hi, I'm called from object reference", 4);

        System.out.println("Nilai retI: " + retI);
    }
}
```

Method Overloading

```
class MyParent {
    public int myMethod(int a, int b) {
        return (a + b);
    }
}

class MyChild extends MyParent {
    public float myMethod(float x, float y) {
        return (x * y);
    }
}

public class MethodOverloadingSample3 {
    public static void main(String args[]) {
        MyParent mp = new MyParent();
        int imp = mp.myMethod(3, 5);
        System.out.println(imp);

        MyChild mc = new MyChild();
        float imc = mc.myMethod(3.2f, 5.1f);
        System.out.println(imc);
    }
}
```

Method Overriding thorugh Inheritance

- Method Overriding:
  - Subclass have a method with the same name and type signatures (return type, argument's types and orders) with a method in its parent
  - The overriding method must not less accessible than the original method
  - The overriding method must not throw any checked exceptions that were not declared for the original method

Method Overriding thorugh Inheritance

```
class OverridingParent {
    public int myMethod(int a, int b) {
        return (a + b);
    }
}

class OverridingChild extends OverridingParent{
    public int myMethod(int x, int y) {
        return (x * y);
    }
}

public class MethodOverridingSample {
    public static void main(String args[]) {
        OverridingParent op = new OverridingParent();
        System.out.println("myMethod on Parent Class: " + op.myMethod(2, 21));

        OverridingChild oc = new OverridingChild();
        System.out.println("myMethod on Child Class: " + oc.myMethod(2, 21));
    }
}
```

Runtime Polymorphism

- Assume that a class named SuperClass defines a method named method.
- Assume that a class named SubClass extends SuperClass and overrides the method named method.
- Assume that a reference to an object of the class named SubClass is assigned to a reference variable named ref of type SuperClass.
- Assume that the method named method is then invoked on the reference variable using the following syntax: ref.method()
- Result: The version of the method named method that will actually be executed is the overridden version in the class named SubClass, and is not the version that is defined in the class named SuperClass.

This is runtime polymorphism, sometimes also referred to as late-binding.

Runtime Polymorphism

- The decision as to which version of the method to execute is based on the actual type of object whose reference is stored in the reference variable, not on the type of the reference variable on which the method is invoked.
- The reason that this type of polymorphism is often referred to as runtime polymorphism is because the decision as to which version of the method to execute cannot be made until runtime. The decision cannot be made at compile time.
- The decision cannot be made at compile time because the compiler has no way of knowing (when the program is compiled) the actual type of the object whose reference will be stored in the reference variable .

Runtime Polymorphism

```
class ParentRP {
    public void myMethod() {
        System.out.println("Hi, from ParentRP");
    }
}

class ChildRP extends ParentRP {
    public void myMethod() {
        System.out.println("Hi, from ChildRP");
    }
}

public class RuntimePolymorphism {
    public static void main(String args[]) {
        ChildRP crp = new ChildRP();
        ParentRP prp = crp;
        prp.myMethod();
    }
}
```

What will be displayed?

Runtime Polymorphism

Now try to add code below to main method, what will be displayed?

```
((ParentRP) prp).myMethod();
```

Runtime Polymorphism

- This is runtime polymorphic behavior.
- The version of the method that was executed was based on the actual type of the object, ChildRP, and not on the type of the reference, ParentRP.

Overview of Interfaces

- An interface definition is similar to a class definition
- Multiple inheritance is allowed when extending interfaces.
- Two kinds of members are allowed in an interface definition:
  - Methods, which are implicitly abstract
  - Variables, which are implicitly constant (final)
- Each interface definition constitutes a new type.
- An interface also cannot be instantiated.
- If a class implements an interface, it must provide a concrete definition for all the methods declared by that interface, and all the methods inherited by that interface.

Overview of Interfaces

- A reference to any object instantiated from any class that implements a given interface can be treated as the type of the interface.
- When a reference to an object is treated as an interface type, any method declared in, or inherited into that interface can be invoked on the reference to the object.
- However, the behavior of the method when invoked on references to different objects of the same interface type may be very different.

Method Overriding thorugh Interfaces

```
interface MyInterfaceA {
     public void m1();
}

interface MyInterfaceB extends MyInterfaceA {
     public void m2();
}

class MyClassA implements MyInterfaceB {
     public void m1() {
          System.out.println("m1 from MyClassA");
     }

     public void m2() {
          System.out.println("m2 from MyClassA");
     }
}

class MyClassB implements MyInterfaceB {
     public void m1() {
          System.out.println("m1 from MyClassB");
     }

     public void m2() {
          System.out.println("m2 from MyClassB");
     }
}
```

Method Overriding thorugh Interfaces

```
public class InterfacePolymorphism {
     public static void main(String args[]) {
          MyInterfaceB mib;

          mib = new MyClassA();
          mib.m1();
          mib.m2();

          mib = new MyClassB();
          mib.m1();
          mib.m2();
     }
}
```

Thank you. Any questions?