Java - Sample Inheritance Problem Set 2

Md. Mohsin Uddin

East West University

mmuddin@ewubd.edu

June 30, 2019

Polymorphism : Example I : Part I

```
class Bank{
    float getRateOfInterest(){return 0;}
}
class SBI extends Bank{
    float getRateOfInterest(){return 8.4f;}
}
class ICICI extends Bank{
    float getRateOfInterest(){return 7.3f;}
}
class AXIS extends Bank{
    float getRateOfInterest(){return 9.7f;}
}
class TestPolymorphism{
    public static void main(String args[]){
        Bank b;
        b=new SBI();
        System.out.println("SBI Rate of Interest: "
                           +b.getRateOfInterest());
        b=new ICICI();
```

Polymorphism : Example I : Part II

```
        System.out.println("ICICI Rate of Interest: "
                                +b.getRateOfInterest());
        b=new  AXIS();
        System.out.println("AXIS Rate of Interest: "
                                +b.getRateOfInterest());
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
SBI Rate of Interest: 8.4
ICICI Rate of Interest: 7.3
AXIS Rate of Interest: 9.7
```

Polymorphism : Example II : Part I

```
class Shape{
    void draw(){
        System.out.println("drawing...");
    }
}
class Rectangle extends Shape{
    void draw(){
        System.out.println("drawing rectangle...");
    }
}
class Circle extends Shape{
    void draw(){
        System.out.println("drawing circle...");
    }
}
class Triangle extends Shape{
    void draw(){
        System.out.println("drawing triangle...");
    }
```

Polymorphism : Example II : Part II

```
}
class TestPolymorphism2{
    public static void main(String args[]){
        Shape s;
        s=new Rectangle();
        s.draw();
        s=new Circle();
        s.draw();
        s=new Triangle();
        s.draw();
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
drawing rectangle...
drawing circle...
drawing triangle...
```

Polymorphism : Example III : Part I

```
class Animal{
    void eat(){
        System.out.println("eating...");
    }
}
class Dog extends Animal{
    void eat(){
        System.out.println("eating bread...");
    }
}
class Cat extends Animal{
    void eat(){
        System.out.println("eating rat...");
    }
}
class Lion extends Animal{
    void eat(){
        System.out.println("eating meat...");
    }
```

Polymorphism : Example III : Part II

```
}
class TestPolymorphism3{
    public static void main(String[] args){
        Animal a;
        a=new Dog();
        a.eat();
        a=new Cat();
        a.eat();
        a=new Lion();
        a.eat();
    }
}
```

When the above code is compiled and executed, it produces the following result:

```
eating bread...
eating rat...
eating meat...
```

Java Runtime Polymorphism with Multilevel Inheritance : Example IV : Part I

```
class Animal{
    void eat(){System.out.println("eating");}
}
class Dog extends Animal{
    void eat(){System.out.println("eating fruits");}
}
class BabyDog extends Dog{
    void eat(){System.out.println("drinking milk");}
    public static void main(String args[]){
        Animal a1,a2,a3;
        a1=new Animal();
        a2=new Dog();
        a3=new BabyDog();
        a1.eat();
        a2.eat();
        a3.eat();
    }
```

Java Runtime Polymorphism with Multilevel Inheritance : Example IV : Part II

```
}
```

When the above code is compiled and executed, it produces the following result:

```
eating
eating fruits
drinking Milk
```

Java Runtime Polymorphism with Multilevel Inheritance : Example V : Part I

```
class Animal{
    void eat(){
        System.out.println("animal is eating...");
    }
}
class Dog extends Animal{
    void eat(){
        System.out.println("dog is eating...");
    }
}
class BabyDog1 extends Dog{
    public static void main(String args[]){
        Animal a=new BabyDog1();
        a.eat();
    }
}
```

Java Runtime Polymorphism with Multilevel Inheritance : Example V : Part II

When the above code is compiled and executed, it produces the following result:

```
Dog is eating
```

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Excercise I : Part I

```
interface GFG {
    void myMethod();
    void getInfo();
}
abstract class Geeks implements GFG {
    void getData() {
        System.out.println("GFG");
    }
}
class Test extends Geeks{
    public void myMethod(){System.out.println("GeeksforGeeks");}
    public void getInfo(){
        System.out.println("Geeks");
    }
    public static void main(String[] args){
        Geeks obj = new Test();
        obj.getInfo();
    }
}
```

Excercise I : Part II

When the above code is compiled and executed, it produces the following result:

```
Geeks
```

References


DEITEL, Java How to Program, 11/e


Java: the complete reference, Herbert Schildt, McGraw-Hill Education Group