Java – Polymorphism

Presented by

Sagar
Java Consultant

1. Method Overloading

- In Java it is legal for a class to have two or more methods with the same name.
-
- Same named methods must be distinguished:
    - by the number of arguments, or
    - by the types of arguments

Advantages of overloading

1.Overloading in Java is the ability to create multiple methods of the same name, but with different parameters.

2.The main advantage of this is cleanliness of code.

3.Method overloading increases the readability of the program.

4.Overloaded methods give programmers the flexibility to call a similar method for different types of data.

5.Overloading is also used on constructors to create new objects given different amounts of data.

Method Overloading…

```
/*
 *  Overloading Example
 */
public class Demo {
   public void test( ) {
      System.out.println("No parameters");
   }

   public void test(int a) {
      System.out.println("a: " + a);
   }

   public void test(int a, int b) {
      System.out.println("a and b: " + a + " " + b);
   }

   public double test(double a) {
      System.out.println("double a: " + a);
      return a * a;
   }
}
```

```
public static void main(String[] args) {
   Demo ob = new Demo( );

   ob.test();


   ob.test(10);

   ob.test(10, 20);

   double result = ob.test(123.2);

   System.out.println("ob.test(123.2): " + result);
}
```

Method Overloading…

- Different result types are insufficient.
  The following will not compile:

```
public double test(double a) {
      System.out.println("double a: " + a);
      return a*a;
}
public int test(double a) {
      System.out.println("double a: " + a);
      return (int) a*a;
}
```

Method Overloading Example

```
public class Demo {
    public void test() {
        System.out.println("No parameters");
    }
    public void test(double d) {
        System.out.println("Inside test(double)");
    }

    public static void main(String[] args) {
        Demo ob = new Demo( );
        ob.test();
        int x = 10;
        ob.test(x);
        double d = 1.2;
        ob.test(d);
    }
}
```

2. Constructor Overloading

```
public class Rectangle {

    private int width;
    private int breadth;


    public void setWidth(int width) {
        this.width = width;
    }

    public void setBreadth(int breadth) {
        this.breadth = breadth;
    }

    public static void main(String[] args) {
        Rectangle rectangle = new Rectangle();


        rectangle.setBreadth(34);
        rectangle.setWidth(12);
    }
}
```

- Rectangle is created at this point in the code, but without breadth and width!*
- You are relying on the Rectangle-user to know that Rectangle creation is a two-part process: one to call the constructor and one to call the setters

Constructor Overloading …

- Using Constructor to initialize state of object
  - The best place to put initialization code is in the constructor. And all you need to do is make a constructor with arguments

```
public class Rectangle {

   private int width;
   private int breadth;

   public Rectangle() {
      width = breadth = 1;
   }

   public Rectangle(int width, int breadth) {
      this.width = width;
      this.breadth = breadth;
   }

   public Rectangle(int size) {
      this.width = size;
      this.breadth = size;
   }
```

No argument constructor

Constructor with arguments

Constructor with arguments

Constructor Overloading …

- Things to remember:
  - A constructor is the code that runs when somebody calls “new” on a class type
    - Rectangle rectangle = new Rectangle(4,5);
  - A *constructor* must have the same name as the class , and no explicit return type
    - public Rectangle(int width, int breadth)
  - If you don't write a constructor in your class, the compiler creates a default *constructor.*
  - You can have more than one constructor in your class, as long as the argument lists are different. Having more than one constructor in a class means you have overloaded constructors

Constructor Overloading …

```
class Circle {
        double radius;
        Circle(double radius) {
            this.radius = radius;
        }
        public double getRadius() {
            return radius;
        }
}
public class Tester {
        public static void main(String[] args) {
                                Circle circle = new Circle();
                                System.out.println(circle.getRadius());
        }
}
```

Constructor Chaining …

- All the constructors in an object's inheritance tree must run when you make a new object.
- Can a Child exist before a parent?
  - The super constructors run to build out the superclass parts of the object first.
- For an object to be fully formed, all the superclass parts must be fully-formed, and that's why the super constructor *must* run.

Constructor Chaining …

- What is the output?

```
class Product {
   public Product() {
      System.out.println("Product");
   }
}

class Mobile extends Product {
   public Mobile() {
      System.out.println("Mobile");
   }
}
```

```
public static void main(String[] args) {
   new Mobile();
}
```

Constructor Chaining …

- Invoking super class constructor using super()

```
public class Product {
   private int productId;
   private String name;
   private double price;

   public Product() {
   }

   public Product(int productId, String name, double price) {
      this.productId = productId;
      this.name = name;
      this.price = price;
   }
```

```
public class Television extends Product {
   private String screenType;
   private String screenSize; // in inches

   public Television() {
   }
   public Television(int productId, String name, double price,
         String screenType, String screenSize) {
      super(productId, name, price);
      this.screenType = screenType;
      this.screenSize = screenSize;
   }
```