PSOOP

Lecture-03

Outline

- Constructor Overloading Concept
- Constructor Overloading Examples
- Method Overloading Concept
- Method Overloading Examples
- Arrays in Java

Constructor Overloading

- We can define multiple constructors within the same class, each with a distinct number/type of parameters.
- This concept is known as ***Constructor Overloading***
- To understand this concept, consider the following example: Suppose you want to buy a pen from a shop. In OOP, this pen is an object. Let us design the Pen class.
- The Pen class can have data members inkColor, brand and price.

Constructor Overloading

```
class Pen{
        String inkColor,brand;
        float price;
}
```

Scenario 1:

You go to the shopkeeper and tell him you want to buy a pen. The shopkeeper would give you the most common pen that people usually buy. This is a default (no-argument) constructor. Let’s add it to the Pen class.

Constructor Overloading

```
class Pen{
        String inkColor,brand;
        float price;
        Pen(){//give me the most common pen
                inkColor="blue";
                brand="Cello";
                price=10;
        }
}
```

Scenario 2:

You go to the shopkeeper and tell him you want to buy a pen with red ink. The shopkeeper would give you the pen with red ink which most people usually buy. This is a parameterized constructor with one parameter. Let's add it to the Pen class.

Constructor Overloading

```
class Pen{
        String inkColor, brand;
        float price;
        Pen(){//give me the most common pen
                inkColor="blue";
                brand="Cello";
                price=10;
        }
        Pen(String inkColor){//give me a red pen (or specific color pen)
                this.inkColor=inkColor;
                brand="Cello";
                price=10;
        }
}
```

Scenario 3:

You go to the shopkeeper and tell him you want to buy a pen with green ink, brand Montex and 20 Rs(better quality). The shopkeeper would give you the pen with required features. This is another parameterized constructor with three parameters. Let's add it to the Pen class.

Constructor Overloading

```
class Pen{
        String inkColor, brand;
        float price;
        Pen(){//give me the most common pen
                inkColor="blue";
                brand="Cello";
                price=10;
        }
        Pen(String inkColor){
        //give me a red pen (or specific color pen)
                this.inkColor=inkColor;
                brand="Cello";
                price=10;
        }
        Pen(String inkColor, String brand, float price){
        //give me a pen with given features
                this.inkColor=inkColor;
                this.brand=brand;
                this.price=price;
        }
}
```

Scenario 4:

Now let's say you go to the shopkeeper with a specific pen and say to the Shopkeeper you want exactly that pen. This is another parameterized constructor with object as parameter. The object is the same as the class. This is called Copy Constructor. You want an exact copy(replica) of the Pen you are having with you. Let's add it to the Pen class.

```
class Pen{
        String inkColor, brand;
        float price;
        Pen(){//give me the most common pen
                inkColor="blue";
                brand="Cello";
                price=10;
        }
        Pen(String inkColor){
        //give me a red pen (or specific color pen)
                this.inkColor=inkColor;
                brand="Cello";
                price=10;
        }
        Pen(String inkColor, String brand, float price){
        //give me a pen with given features
                this.inkColor=inkColor;
                this.brand=brand;
                this.price=price;
        }
        Pen(Pen p){//give me an exact replica of this pen
                inkColor=p.inkColor;
                brand=p.brand;
                price=p.price;
        }
        void display(){
        System.out.println("Color="+inkColor+"\nBrand="+brand+"\nPrice="+price);
        }
}
```

The class we designed is a class which contains overloaded constructors and also a display() method. Lets create objects of this class and test it from main()

Class TestPen

```
class TestPen{
        public static void main(String []args){
                Pen p1=new Pen();//no argument constructor call
                Pen p2=new Pen("red");//parameterized constructor 1 calle
                Pen p3=new Pen("green","Montex",20);//parameterized co
                Pen p4=new Pen(p3);//copy constructor
                p1.display();
                p2.display();
                p3.display();
                p4.display();
        }
}
```

```
Color=blue
Brand=Cello
Price=10.0
Color=red
Brand=Cello
Price=10.0
Color=green
Brand=Montex
Price=20.0
Color=green
Brand=Montex
Price=20.0
```

Method Overloading Concept

- ✓ Defining multiple methods with the same name but different signatures within the same class
- ✓ Java compiler determines which method is used based on the method signature
- ✓ Overloaded methods must have different parameter lists. You cannot overload methods based on different modifiers or return types

Method Overloading Example

```
class Shape{
    void area(float r){//Circle
        float ar=r*r*(float)Math.PI;
        System.out.println("Area="+ar);
    }
    void area(float l,float b){//Rectangle
        float ar=l*b;
        System.out.println("Area="+ar);
    }
    void area(float s1,float s2,float h){//trapezium
        float ar=0.5f*(s1+s2)*h;
        System.out.println("Area="+ar);
    }
}
```

Method Overloading Example contd..

```
class TestShape{
    public static void main(String a[]){
        Shape a1=new Shape();
        a1.area(3.5f);//a1 is a circle
        a1.area(4.67f,6.7f);//a1 is a rectangle
        a1.area(5.5f,7f,8.7f);//a1 is a square
    }
}
```

```
C:\NSM\PSOOP 2023-24\programs>java TestShape
Area=38.484512
Area=31.289
Area=54.375
```

What is the output?

```
class Test{
    double test(double a){
        return a;
    }
    int test(double a){
        return (int)a;
    }
}
class Main{
    public static void main(String []arr){
        System.out.println(new Test().test(3));
    }
}
```

Will this code compile?

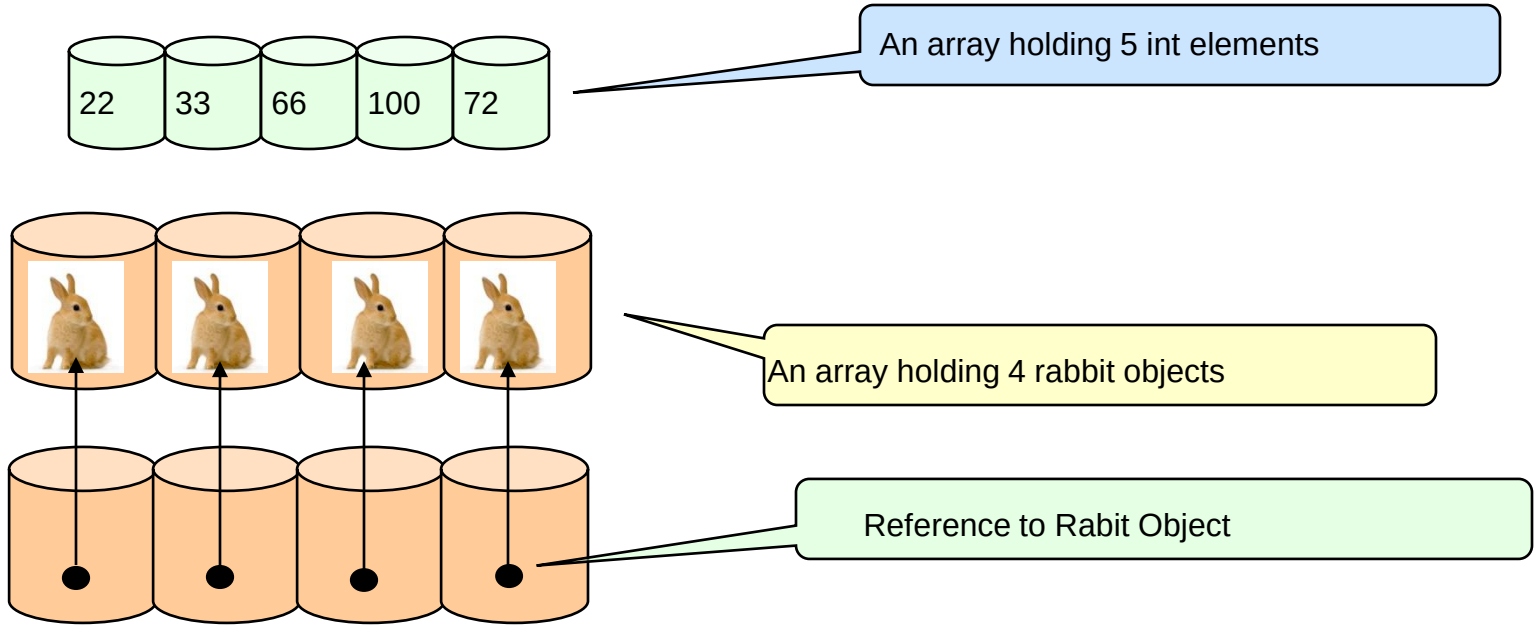
```
class Test{
    double test(double a){
        return a;
    }
    int test(int a){
        return (int)a;
    }
}
class Main{
    public static void main(String []arr){
        System.out.println(new Test().test(3));
        System.out.println(new Test().test(6.5));
    }
}
```

What is the output of this code?

```
class Test{
    double test(int a,int b){
        System.out.println("first");
        return a+b;
    }
    double test(double a,double b){
        System.out.println("second");
        return a+b;
    }
    double test(float a,float b){
        System.out.println("third");
        return a+b;
    }
}
class Main{
    public static void main(String []arr){
        System.out.println(new Test().test(4,5));
        System.out.println(new Test().test(4f,5f));
        System.out.println(new Test().test(4f,5.0));
    }
}
```

Arrays in Java

- A data structure which defines an ordered collection of a fixed number of homogeneous data elements
- Size is fixed and cannot increase to accommodate more elements
- Arrays in Java are objects and can be of primitive data types or reference variable type
- All elements in the array must be of the same data type



Arrays in Java (Contd…)

- *Reference variables* are used in Java to store the references of objects created by the operator *new*
- Any one of the following syntax can be used to create a reference to an *int* array

```
int x[];
int [] x;
```

- The reference x can be used for referring to any *int* array

```
//Declare a reference to an int array
int [] x;
//Create a new int array and make x refer to it
x = new int[5];
```

Arrays in Java (Contd…)

- The following statement also creates a new *int* array and assigns its reference to x

```
int [] x = new int[5];
```

- In simple terms, references can be seen as names of an array

Initializing Arrays

- An array can be initialized while it is created as follows:

```
int [] x = {1, 2, 3, 4};

char [] c = {'a', 'b', 'c'};
```

Length of an Array

- Unlike C, Java checks the boundary of an array while accessing an element in it
- Programmer not allowed to exceed its boundary
- And so, setting a for loop as follows is very common:

```
for(int i = 0; i < x.length; ++i){
        x[i] = 5;
}
```

This works for any size array

use the .length attribute of an array to control the *for* loop

Example

```
public class TestArray {
   public static void main(String[] args) {
      double[] myList = {1.9, 2.9, 3.4, 3.5};
      // Print all the array elements
      for (double element: myList) {//for each loop
         System.out.println(element);
      }
   }
}
```

Otput:

```
1.9
2.9
3.4
3.5
```

Can we change the array length?

- int[] primes=new int[10];

  ……

  primes=new int[50];
- Previous array will be discarded

2D Arrays

- Representing 2D arrays
  - int myArray[][];
  - myArray = new int[3][4];
  - int myArray [][] = new int[3][4];
- Example
- int myarray[][]={{0,0,0},{1,1,1}};

2 columns and 3 rows

Multidimensional Arrays

- A Multi-dimensional array is an array of arrays

- To declare a multidimensional array, specify each additional index using another set of square brackets
- Refer example: ArrayExample.java

```
int [][] x;
//x is a reference to an array of int arrays
x = new int[3][4];
//Create 3 new int arrays, each having 4 elements
//x[0] refers to the first int array, x[1] to the second and
so on
//x[0][0] is the first element of the first array
//x.length will be 3
//x[0].length, x[1].length and x[2].length will be 4
```

THANK YOU