OBJECT ORIENTED PROGRAMMING

Affefah Qureshi

Department of Computer Science

Iqra University, Islamabad Campus.

OVERLOADING METHODS

- In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different.
- When this is the case, the methods are said to be *overloaded*, *and the process is referred to as method overloading*.
- *Method* overloading is one of the ways that Java supports polymorphism.
- When an overloaded method is invoked, Java uses the type and/or number of arguments as its guide to determine which version of the overloaded method to actually call.
- Thus, overloaded methods must differ in the type and/or number of their parameters.
- While overloaded methods may have different return types, the return type alone is insufficient to distinguish two versions of a method.

METHOD OVERLOADING

- Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different.
- It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.
- Three ways to overload a method

1. Number of parameters.
   - add(int, int)
   - add(int, int, int)
2. Data type of parameters.
   - add(int, int)
   - add(int, float)
3. Sequence of Data type of parameters.
   add(int, float)
   - add(float, int)

- Invalid case of method overloading:
  - int add(int, int)
  - float add(int, int)

EXAMPLE

```
//Demonstrate method overloading.
class OverloadDemo {
            void test() {
                        System.out.println("No parameters"); }
//Overload test for one integer parameter.
            void test(int a) {
                        System.out.println("a: " + a); }
//Overload test for two integer parameters.
            void test(int a, int b) {
                        System.out.println("a and b: " + a + " " + b); }
//overload test for a double parameter
            double test(double a) {
                        System.out.println("double a: " + a); return a*a; }
}
class Overload {
            public static void main(String args[]) {
                        OverloadDemo ob = new OverloadDemo(); double result;
//call all versions of test()
                        ob.test();
                        ob.test(10);
                        ob.test(10, 20);
                        result = ob.test(123.25);
                        System.out.println("Result of ob.test(123.25): “+ result); } }
```

Output:

```
No parameters
a: 10
a and b: 10 20
double a: 123.25
Result of ob.test(123.25): 15160.0625
```

EXAMPLE

```
Output:

Volume of mybox1 is 3000.0
Volume of mybox2 is -1.0
Volume of mycube is 343.0
```

```
// Constructor Overloading
class Box {
   double width;
   double height;
   double depth;

   // Constructor with three parameters
   Box(double w, double h, double d) {
      width = w;
      height = h;
      depth = d;
   }

   // Default constructor
   Box() {
      width = -1;
      height = -1;
      depth = -1;
   }

   // Constructor with one parameter (for creating a cube)
   Box(double len) {
      width = height = depth = len;
   }

   // Method to calculate the volume of the box
   double volume() {
      return width * height * depth;
   }
}

class OverloadCons {
   public static void main(String args[]) {
      // Creating boxes using the various constructors
      Box mybox1 = new Box(10, 20, 15);
// Box with dimensions 10x20x15
      Box mybox2 = new Box();
 // Default box with uninitialized dimensions
      Box mycube = new Box(7);
 // Cube with dimensions 7x7x7
      double vol;

      // Calculate and print the volume of each box
      vol = mybox1.volume();
      System.out.println("Volume of mybox1 is " + vol);

      vol = mybox2.volume();
      System.out.println("Volume of mybox2 is " + vol);

      vol = mycube.volume();
      System.out.println("Volume of mycube is " + vol);
   }
}
```