Atma Ram Sanatan Dharma College
University of Delhi

Programming in Java

Practical Assignment for Paper Code 32341201

Submitted By
Sudipto Ghosh
College Roll No. 19/78003
BSc (Hons) Computer Science
Part I Semester II

Submitted To
Dr Parul Jain
Department of Computer Science

Question 1

What is the difference between function overloading and constructor overloading in Java?

Write a suitable program to illustrate the following:

(a) Default constructor
(b) Parameterized constructor
(c) Method overloading with different number of parameters
(d) Method overloading with type of parameters
(e) Method overloading with different sequence of parameters

Answer

(i) **Function Overloading:** In Java, when we have two or more methods or functions with the same name but with different formal parameter lists which can differ in types, number and order of parameters. This is called function or method overloading and is an example of static polymorphism. The appropriate function to be called is resolved statically on the basis of actual parameter lists.

Constructor Overloading: Constructors are class methods with the same name as that of the class and are used to initialize the data members (state) of an object. Similar to function overloading, we can have multiple definitions of a constructor with different formal parameter lists. If we don't define any constructor, the compiler creates the default constructor by default during compilation. If we have defined a parameterized constructor, then compiler will not create default constructor and vice versa. The appropriate constructor is called when the object is instantiated on the basis of the actual parameter list.

(ii) **Source Code**:

```
class Example {
  // Private Data Member
  private int val;
  // Default Constructor
  Example() {
    val = 0;
  }
  // Paramterized Constructor
  Example(int val) {
    this.val = val;
  }
  // Getter Method
  int getVal() {
    return val;
  }
  // Example Method
  int method(int n) {
    return n;
  }
  // Method Overloading with different number of parameters
  int method(int n, int m) {
    return n + m;
  }
  // Method Overloading with type of parameters
  int method(char c) {
    return (int) (c);
  }
  // Method Overloading with different sequence of parameters
  int method(char c, int m) {
    return (int) (c) + m;
  }
```

```
  int method(int m, char c) {
    return (int) (c) - m;
  }
  // Driver Code
  public static void main(String[] args) {
    Example e1 = new Example();
    Example e2 = new Example(2);
    System.out.println(e1.getVal());
    System.out.println(e2.getVal());
    System.out.println(e1.method(10));
    System.out.println(e1.method(10, 29));
    System.out.println(e1.method('A'));
    System.out.println(e1.method('B', 10));
    System.out.println(e1.method(10, 'C'));
  }
}
```

Output:

0
2
10
39
65
76
57

Question 2

Write the advantages of using packages in Java. Write a suitable program that illustrates different levels of protection in classes/subclasses belonging to same package or different packages.

Answer

Advantages of using Packages: Packages are name spaces and are used to organize related classes, interfaces and other sub packages.

- It prevents naming collisions for classes and interfaces which might have the same name but are implemented for a different context. For example: `Date` class is available as `java.util.Date` and `java.sql.Date` and are to be used for different use cases.
- It makes searching, locating, using and documenting code easier as the classes and interfaces are organized according to a defined folder structure.
- Packages can be used to provide granular access control to interfaces, classes, methods and data members. The default and `protected` access specifiers make a member accessible by classes in the same package and classes and subclasses belonging to the same package respectively.

Program to Illustrate Levels of Protection

Source Code

```
/** package1/Example.java **/
package package1;
import package2.*;
class Example {
  public static void main(String[] args) {
    A a = new A();
    a.printpubA(); // works
    // a.printproA(); // A::printproA() not accessible to package1.Example
```

```
    // a.printpriA(); // A::printpriA() not accessible to package1.Example
    // a.printdefA(); // A::printdefA() not accessible to package1.Example
    // B b = new B(); // class B is not accessible to package1.Example
    // b.printdefB(); // B::printdefB() not accessible to package1.Example
    a.printpubB(); // works
  }
}

/** package2/A.java **/
package package2;
public class A {
  public void printpubA() {
    printproA(); // works
  }
  protected void printproA() {
    printdefA(); // works
  }
  private void printpriA() {
    System.out.println('A');
  }
  void printdefA() {
    printpriA(); // works
  }
  public void printpubB() {
    B b = new B(); // works
    b.printdefB(); // works
  }
}

/** package2/B.java **/
package package2;
class B {
  void printdefB() {
    System.out.println('B');
  }
  public static void main(String[] args) {
    A a = new A(); // works
    a.printpubA(); // works
    a.printproA(); // works
    // a.printpriA(); // A::printpriA() not accessible to package2.B
    a.printdefA(); // works
  }
}
```

Output

- Compilation: `javac package1\Example.java`
  `javac package2\B.java`
- On running `java package1.Example` we get,

  A
  B

- On running `java package2.B` we get,

  A
  A
  A