UNIT 4: CLASS IN JAVA

UNIT STRUCTURE

4.1 Learning Objectives

4.2 Introduction

4.3 Class Fundamentals

4.4 Accessing Class Members

4.5 Declaring Objects

4.6 Passing Arguments to Method

4.7 Returning Multiple Values from Methods

4.8 Java Modifiers

4.9 Let Us Sum Up

4.10 Further Reading

4.11 Answer to Check Your Progress

4.12 Model Questions

4.1 LEARNING OBJECTIVES

After going through this unit, you will be able to:

- describe Java class
- define about the class members
- explain the process of class object declaration
- describe how to return the multiple values from Java methods.

4.2 INTRODUCTION

In the earlier units, we discussed the basic programming concepts of Java. Topics covering basic elements like tokens, keywords, constants of Java programming language were discussed in the previous unit. The previous unit also includes topics like how to declare variables in Java, different data types, operators, input/output statements, and about scope and lifetime of variables. Loop control statements with their types were also discussed in the previous unit.

In this unit, you will be introduced to class fundamentals in Java. Topics like accessing class members and declaring objects are discussed in detail in this unit. In addition to these, concepts related to passing arguments to methods and returning multiple values from methods are also described in this unit. In the next unit, we will explore the topics related to OOP in java.

4.3 CLASS FUNDAMENTALS

A Java program is represented by a class that contains a block of codes and statements. It is a user-defined data type with syntax to define its properties.

You can define the Java class as follows:

Syntax :

```
class class-name
{
        datatype      variable_1;
        datatype instance-variable_2;
        -
        -
        datatype   variable_n;
        datatype method_name1(parameter-list)
        {
        //method body
        }
        -
        -
        datatype method_name(parameter-list)
        {
        //method body
        }
}
```

Instance variables are data or variables defined within a class. Methods contain the codes. Members of the class are the collection of

methods and variables present in a class.

Example:
```
class shelf
{
      double width, height, depth;
      double volume()
      {
            return width*height*depth;
      }
}
```

4.4 ACCESSING CLASS MEMBERS

In Java, all class members are accessed by the "dot" operator.

The syntax for accessing the instance variable of the class is :

object name.variable name;

where object name is the object and variable name is the instance variable.

Example:
```
rectangle2.length= 10;
rectangle2.width= 8;
```

The syntax for accessing instance methods of the class is :

objectname.methodname(parameter-list);

where, objectname is the object, methodname is the instance method and parameter-list is a list of comma separated values.

Example:
```
void getdata(int a, int b) {
      length=a;
      width=b;
}
rect1.getdata(8,10);
```

4.5 DECLARING OBJECTS

An object is declared by defining a variable with its data type and name.

Syntax: Class ClassName;

ClassName object = new ClassName();

Example:
```
Square squ1;
squ1=new Square();
```

4.6 PASSING ARGUMENTS TO METHODS

Arguments are passed to the method through two categories:

- **Call by value:** The value of an augment is copied to the formal parameter of the sub-routine. The sub-routine does not affect the argument used to call it if any changes are made to the parameter.

Example:

```
class test
{
      void meth(int x, int y)
      {
            x*=2;
            y/=2;
      }
}
class abc
{
      public static void main(String[]args)
      {
            test  obj=new test();
            int i-4,j=7;
            system.out.println("Before
             calling  i="+i+"j="+j);
            obj.meth(i,j);
            system.out.println("After
            calling i="+i+"j="+j");
      }
      }
```

Output: Before calling i=4 j=7

After calling i=4 j=7

Call by reference: A reference to an argument is passed to the parameter. Any changes made to the parameter affect the argument used to call it because this reference is used to access the actual argument identified in the call.

Example:

```
class test
{
      int a,b;
      test (int i,int j)
      {
            a=i;
            b=j;
      }
      void meth(test ox)
      {
            ox.a*2;
            ox.b/2;
      }
}
class abc
{
    public static void main(String[]args)
   {
      test obj=new test (4,5);
      system.out.println("Before Calling a= "+
      obj.a+ "b="+ obj.b );
      obj.meth(obj);
       system.out.println("After calling a="obj.a
      + "b="+ obj.b" );
   }
}
```

Output: Before calling a=4 b=5
After calling a=8 b=2

4.7 RETURNING MULTIPLE VALUES FROM METHODS

In Java, the methods can return only one value at a time. Since the returning of multiple values from a method is not possible we use alternatives to return multiple values in Java. Five ways to return multiple values from methods are:

- **Using a POJO class instance:** Using a POJO class instance means to return an instance of a class consisting of all fields to be returned. It is the most common method used to return multiple values. This method provides type-safety and also improves the readability of the program. This class can be a Plain Old Java Object (POJO) with public member variables and a public constructor to begin the fields with corresponding values or a JavaBean for their private member variables.

  Example:

```
// POJO for grouping multiple fields
final class People
{
    public String name;
    public int age;
    public char gender;
  public Person(String name, int age, char gender)
   {
        this.name = name;
        this.age = age;
        this.gender = gender;
  }
}
class Main
{
    public static Person getDetails()
```

```
{
    // return person details from the method
    String name = "Jane";
    int age = 12;
    char gender = 'F';
    return new Person(name, age, gender);
}
 // Return multiple values from a method in Java
public static void main(String[] args)
{
   Person person = getDetails();
   System.out.println("Name is " + person.name);
   System.out.println("Age is " + person.age);
  System.out.println("Gender is " + person.gender);
}
}
```

Output: Name is Jane

Age is 12

Gender is F

- **Using javafx.util.Pair class:** The Pair class included in the javafx.util package representing the name-value pairs can be used for returning multiple values from a method from Java 8 onwards.

  Example:

```
import javafx.util.Pair;
class Main
{
    public static Pair[] getDetails()
    {
    return new Pair[] { new Pair<>("name", "Jane"),
    new Pair<>("age", 12),
    new Pair<>("gender", 'F') };
    }
```

```
// Return multiple values from a method in
//Java 8 and above
 public static void main(String[] args)
 {
      Pair[] person = getDetails();
      for (int i = 0; i < person.length; i++)
      {
      System.out.println("{" + person[i].getKey()
       + " : " +
       person[i].getValue() + "}");
      }
    }
}
```

Output: {name : Jane}
{age : 12}
{gender : F}

- **Return an array of specific type or an object array:** Returning an array is the elementary approach to return multiple values but it neither offers type-safety for an object array nor passes field information to the caller.

  Example:

```
import java.util.Arrays;
class Main
{
    public static int[] getDetails()
    {
        int v1 = 35;
        int v2 = 55;
        int v3 = 95;
       eturn new int[] { v1, v2, v3 };
    }
  // Return multiple values from a method in Java
    public static void main(String[] args)
```

```
    {
          int[] ints = getDetails();
        System.out.println(Arrays.toString(ints));
    }
}
```

Output: [35, 55, 95]

- **Return a collection:** Instead of returning an array a Collection can be returned, like a List.

  Example:

```
import java.util.Arrays;
import java.util.List;
class Main
{
    public static List<Object> getDetails()
    {
        String name = "Jane";
        int age = 25;
        char gender = 'F';
        long income = 100000;
         return Arrays.asList(name, age,
        gender, income);
    }
   // Return multiple values from a method in Java
    public static void main(String[] args)
    {
        List<Object> person = getDetails();
        System.out.println(person);
    }
}
```

Output: [Jane, 25, F, 100000]

- **Using a delimiter:** A string having all the values separated by a delimiter is returned and after returning the string, it is separated using the same delimiter before the values are used inside the call method.

Example:
```
class Main{
    public static String getDetails() {
        return "Jane" + "~" + "Mike";
    }
   // Return multiple values from a method in Java
    public static void main(String[] args)    {
        String[] name = getDetails().split("~");
        for (String s: name) {
            System.out.println(s);
        }
  }
}
```

Output: Jane
Mike



CHECK YOUR PROGRESS

Q 1: What is the default value of the local variables?

.....

Q 2: What are the various access specifiers in Java?

.....

4.8 JAVA MODIFIERS

While declaring a method in a Java program, there is an optional part in the syntax for modifiers. Modifiers are used in method declaration. There are a number of modifiers, they are:

- **public, protected, default or private :** These four modifiers define the scope,what class can invoke which method. They are also called visibility modifiers. They can be one of these values.
- **static :** This modifier is used for declaring class methods and variables. The method can be invoked on the class without creating an instance of the class.
- **abstract :** The abstract method must be overridden in the subclass

and the class should be extended.

- **final :** An overridden method cannot be there in a subclass.
- **native :** Another language is used to implement the method.
- **synchronized :** Only one thread at a time can access the methods.
- **throws :** This method throws a list of exceptions.



CHECK YOUR PROGRESS

Q 3: What is an object?

.....

Q 4: What is object-oriented paradigm?

.....

Q 5: What is a return a collection?

.....

Q 6: What is POJO class instance?

.....



4.9 LET US SUM UP

- A Java program is represented by a class that contains a block of codes and statements.
- Instance variables are the data or variables defined within a class. Methods contain the codes. Members of class is the collection of methods and variables present in a class.
- In Java, all class members are accessed by the "dot" operator. Syntax for accessing instance variable of the class is : objectname. variablename;
- An object is declared by defining a variable with its data type and name.
  Syntax:
  Class ClassName;
  ClassName object = new ClassName();
- **Call by value:** The value of an argument is copied to the formal parameter of the sub-routine. The sub-routine does not affect the argument used to call it if any changes are made to the parameter.

- **Call by reference:** A reference to an argument is passed to the parameter. Any changes made to the parameter affect the argument used to call it because this reference is used to access the actual argument identified in the call.
- Using a POJO class instance means to return an instance of a class consisting of all fields to be returned.
- The Pair class is included in the javafx.util package representing the name-value pairs can be used for returning multiple values from a method from Java 8 onwards.
- Returning an array is the elementary approach to return multiple values but it neither offers type-safety for an object array nor passes field information to the caller.
- Instead of returning an array, a Collection can be returned, like a List.
  Example: import java.util.Arrays;
  import java.util.List;
- While declaring a method in a Java program, there is an optional part in the syntax for modifiers. Modifiers are used in method declaration.



4.10 FURTHER READINGS

1) Schildt, H., & Coward, D. (2014). Java: the complete reference (p. 1312). New York: McGraw-Hill Education.



4.11 ANSWERS TO CHECK YOUR PROGRESS

Ans to Q No 1: The local variables are not initialized to any default value, neither primitives nor object references.

Ans to Q No 2: In Java, access specifiers are the keywords that are used to define the access scope of the method, class, or variable. In Java, there are four access specifiers given below.

- Public The classes, methods, or variables which are defined as public, can be accessed by any class or method.
- Protected can be accessed by the class of the same package, by the sub-class of this class, or within the same class.

- Defaults are accessible within the package only. By default, all the classes, methods, and variables are of default scope.
- Private The private class, methods, or variables defined as private can be accessed within the class only.

Ans to Q No 3: The Object is the real-time entity having some state and behavior. In Java, an Object is an instance of the class having the instance variables as the state of the object and the methods as the behavior of the object. The object of a class can be created by using the new keyword.

Ans to Q No 4: It is a programming paradigm based on objects having data and methods defined in the class to which it belongs. The object-oriented paradigm aims to incorporate the advantages of modularity and reusability. Objects are the instances of classes that interact with one another to design applications and programs.

Ans to Q No 5: Instead of returning an array, a collection can be returned, like a list. Example: import java.util.Arrays;
import java.util.List;

Ans to Q No 6: Using a POJO class instance means to return an instance of a class consisting of all fields to be returned. It is the most common method used to return multiple values. This method provides type-safety and also improves the readability of the program. This class can be a Plain Old Java Object (POJO) with public member variables and a public constructor to begin the fields with corresponding values or a JavaBean for their private member variables.



4.12 MODEL QUESTIONS

Q 1: What is object-oriented paradigm?

Q 2: What is the difference between an object-oriented programming language and object-based programming language?

Q 3: What is the purpose of static methods and variables?

Q 4: What are the advantages of Packages in Java?

Q 5: What if I write static public void instead of the public static void?

*** ***** ***