UNIT 1: OBJECTS AND CLASSES

Unit Structure

1.0 Learning Objectives

After learning this unit, you will be able to:

- Discuss the general form of class.
- Describe argument passing.
- Define constructors.
- Explain the ‘this’ keyword and finalise () method.

.

1.1 Introduction

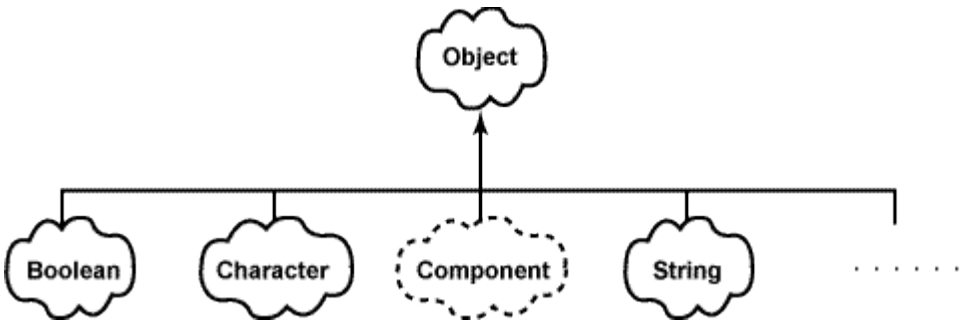
"A class is nothing but a blueprint or a template for creating different objects which defines its properties and behaviors. Java class objects exhibit the properties and behaviors defined by its class. A class can contain fields and methods to describe the behavior of an object."

"Methods are nothing but members of a class that provide a service for an object or perform some business logic. Java fields and member functions names are case sensitive. Current states of a class's corresponding object are stored in the object's instance variables. Methods define the operations that can be performed in java programming."

1.2 The General Form of a class

Object Oriented Concepts:

An object is an identifiable entity with some characteristic features and behavior. Anything which has some properties and performs some behavior is called an object.



For Example:

Pen is an object, as its characteristic features are its shape, colour etc. that is, it's cylindrical in shape and blue in colour and the characteristic behavior is the purpose of using it, that is, it is used for writing.

Consider another Example: Chair, Table, Eraser, Note Book all are objects, as all of them have some characteristic feature and perform some behavior.

Object, Classes and Features

Classes:

Collection of similar types of objects is called Class. A Class is also called as an Object Factory as once the class is created we can create as many objects as we wish using that class.

The concept of Class will become clearer with the help of this Example: Consider that in a school, in drawing class, the teacher has a sample copy of a card (can be used for birthdays, anniversary etc.) which has to be made by each and every student. Now, as the teacher is having the sample copy with her which will be showed to the students for reference so that students can also make the same type of card with the same length and breadth.

Using that sample copy a number of cards with the same measurement, which can be used for different purposes, will be created by the students. Putting it differently, once the sample copy is created i.e. Class, it can be used as a reference in creating number of copies i.e. Object.

The sample copy cannot be used, as it is the copy, which will only show the exact measurement for reference of creating another card.

A collection of statements compiled together in order to execute an operation is defined as a Java method. For Example: the system executes a collection of statements to display a message while performing the System. Out print In method.

Given below is the technique of creating your own method with or without values, how to invoke a method with or without parameters, how to apply method abstraction in the program design and how to overload methods using same names.

Creating a Method:

In general, a method has the following syntax:

Modifier return value type method name (list of parameters)

{

//Method Body;

}

A method definition consists of a method header and a method body. All the parts comprising a method are listed below:

1. **Modifiers -** It tells the compiler how the method is to be called and defines the access type of the method. The modifier is optional.
2. **Return Type -** The method may or may not return a value. In case it does, the return Value Type is the data type of the value which the method returns. Some methods perform the desired operations without returning a value. In this case, the return Value Type is the keyword void.
3. **Method Name -** This refers to the actual name of the method, which along with the parameter list comprises the method signature.
4. **Parameters -** A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order and number of the parameters of a method. Parameters are optional; i.e., a method may contain no parameters."
5. **Method Body** - This is a collection of statements that define what the method does.

```
class display
{
privateint x;
voidgetdata(int a)
{
x=a
}
intreturndata( )
{
return x;
}
}
```

In the above program, display is the name of the class with one data member, x and member functions, get data ( ) and return data ( ).

Creating Objects

Once the above code is written, a class with display name gets created along with the specified data members and member functions. However, no more objects are created at this point of time. In order to create objects, the new operator is used. For Example:

classnameobjectname= new classname( )

It can also be alternatively written as:

Classnameobjectname;

Objectname=new classname ( );

The new operator dynamically allocates memory for an object and returns a reference to it. The default value of object data type is null.

For Example:

display d; //creates a reference

d=new display ( );

Now let us write a program to show the method of class creation and the way objects are created. For Example:

class display

{

```
int x;
int y;

void getdata1(int a)
{
x=a;
}

int returndata1( )
{
return x;
}

void getdata2(int b)
{
y=b;
}

int returndata2( )
{
return y;
}
}

class check
{

public static void main (String[] args)
```

```
{

display  c= new display( );
c.getdata1 (10);
c.getdata2 (20);
System.out.println ("Value of x" + c.returndata1 ( ))
System.out.println ("Value of y" + c.returndata2 ( ))
}

}
```

Check your progress 1

1. Define a class.
2. Write the syntax of a method.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

1.3 Argument Passing

Generally, there are two different methods of passing an argument to a function. These methods are:

a. Call by value

b. Call by reference

a. **Call by value -** In call by value method, the value of an argument is copied to the formal parameter. That is, the changes made in the actual argument are not reflected into the formal parameter.

Let us consider an Example: to illustrate the same:

//Call by Value Method

```
class check
{
inta,b;
void add(int x, int y)
{

a=x+10;
b=y+20;
}

}
class display

{
```

```
public static void main(String args [])

{

check c= new check ( );
int a=15, b=30;
System.out.println (“a and b before call” + a + “ “ + b);
c.add(a,b);
System.out.println (“a and b after call” + a + “ “ + b);
}

}
```

The output of above program will be:

a and b before call 15 30

a and b after call 15 30

b. Call by Reference - In the Call by Reference method, the changes made to the actual argument are also reflected in the formal argument. Let us take an Example: to see the illustration of the same:

```
//Call by Reference
class check
{

int a, b;

check (inti, int j)
```

```
{
a=i;
b=j;
}

//pass an object

void add (check c)
{

c.a+=5;
c.b*=4;
}

}

class display

{

public static void main (String args [])

{

check c= new check (4, 8);
System.out.println (“c.a and c.b before call: “+ c.a + “ “+ c.b);
c.add(c);
System.out.println (“c.a and c.b after call: “+ c.a + “ “ + c.b);
```

```
}

}
```

The output of the above program will be:

```
c.a and c.b before call: 4,8
c.a and c.b after call: 9,32
```

Notice the output of the above program, the values of a and b before and after the function call are different as the reference of these variables are passed to formal parameters. Hence, the changes made in actual parameters are directly reflected in formal parameters.

Check your progress 2

1. Explain the call by value method.
2. Write a note on the call by reference method.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

1.4 Constructors

Till now whatever we have studied, we have seen that if you want to initialize a variable it has to be done at the time of object creation. But it can be a tedious job to initialize all the variables in a class each time an object is created.

In the above program, the set ( ) function is used to initialize the variables a and b, but, what if, the programmer forgets to call this set ( ) function in the main ( ) function. Then, it may happen that variables a and b can take garbage values.

So, in spite of giving the job of initialization to the programmer the entire responsibility of initialization is given to the compiler, which in turn is achieved by using constructors.

The constructors are not called in the main ( ) function. They are automatically executed at the time of object creation. For Example:, the above program using default constructors can be written as:

Default Constructor

The default constructors are those constructors which do not accept any parameters/arguments. These constructors are automatically executed at the time of object creation.

Let us take an Example: to illustrate the same:

```
Class rectangle
{
double length;
double breadth;
//Constructor for rectangle
rectangle ( )
{
System.out.println (“Constructing Rectangle”);
length= 5.0;
breadth= 8.0;
}
```

```
double area( )
{
return length * breadth;
}
}

Class demo
{
public static void main (String args[])
{
//creating objects
rectangle r1 = new rectangle ( );
double result;
//get area of rectangle
result= r1.area( );
System.out.println (“Area is” + result);
//get area of second box
}
}
```

The output of above program is given below:

```
Constructing Rectangle
Area is 40
Area is 40
```

Notice the execution of the above program. At the time of object creation, the default constructors get executed, which can be visualized by the statement “Constructing Rectangle” which gets displayed when the object is created.

Parameterised Constructors

The parameterised constructors are those constructors which accept parameters/arguments. At the time of object creation the values of these arguments/parameters are passed from main ( ) function.

Consider the given Example:

```
class rectangle
{
double length;
double breadth;
//Constructor for rectangle
rectangle (double x, double y)
{
length =x;
breadth =y;
}
//compute and return area
double area ( )
{
return length * breadth;
}

}

class demo
{
public static void main (String args[])
{
//declare, allocate and initialize rectangle objects
```

```
rectangle r1= new rectangle(3,5);
rectangle r2=new rectangle (10,20);
double result;
//get area of first box
result=r1.area ( );
System.out.println ("Area is" + result);
//get area of second box
result=r2.area ( );
System.out.println ("Area is" + result);
}
}The output of above program will be:
Area is 15.0
Area is 200.0
```

In the above program, the values to the parameterised constructors are passed at the time of object creation in main function.

Check your progress 3

1. Write a note on default constructor.
2. Write an Example: for parameterised constructor.

.....

.....

.....

.....

.....

.....

.....

.....

1.5 The ‘this’ keyword

The ‘this’ keyword is used when a function will need to refer to the object which invoked it. The ‘this’ keyword can be used inside any method to refer to the current object. That is, it is always a reference to the object on which the method was invoked. Let us consider an Example: to illustrate the same:

```
Rectangle (double length, double breadth)
{
this.length=length;
this.breadth=breadth;
}
```

The use of this keyword is redundant but perfectly correct. Inside rectangle ( ), this will always refer to the invoking object. When a local variable has the same name as an instance variable, the local variable hides the instance variable. That is why length and width were not used as the name of the parameters to the rectangle ( ) constructor inside the rectangle class. If we would have used in that way, then breadth would have referred to the formal parameter hiding the instance variable breadth

Check your progress 4

1. Explain the use of this keyword.
2. Where can this keyword be used?

.....

.....

.....

.....

.....

.....

.....

.....

1.6 The Finalise ( ) Method

Till now, we have studied about the process of object creation. When objects are created using new keyword, the matching constructor is called automatically, where we initialize members of the class. This process is called initialization.

Sometimes an object needs to perform some operation when it is destroyed. For Example:, if an object is holding some non-Java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, Java provides a mechanism called finalization.

Using the finalization keyword, you can define specific actions which will occur when an object is just to be reclaimed by the garbage collector. To add a finalizer to a class, simply define the finalise ( ) method, the Java run time calls that method whenever it is about to recycle an object of that class.

```
protected void finalize ( )
{

//finalisation code here

}
```

As you know that Java's garbage collector runs in its own thread, it will run transparently alongside the execution of the program. If the programmer explicitly wants to request a garbage collection at some point of time, then it can be done by invoking System.gc( ) or Runtime.gc( ), which will fire off garbage collection at that time

Check your progress 5

1. Explain initialisation.
2. Write how to add a finaliser class.

.....

.....

.....

.....

.....

.....

.....

.....

.....

1.7 Let Us Sum Up

In this Unit we have learned Object Oriented Concepts we can say that a object is an identifiable entity with some characteristic features and behavior. Anything which has some properties and performs some behavior is called an object. Keeping mind the programming for user's application is said as object oriented programming. In mean time we understood Classes which is nothing but a Collection of similar types of objects is called Class. A Class is also called as an Object Factory as once the class is created we can create as many objects as we wish using that class.

There is something called Java Method .A Java method is a collection of statements that are grouped together to perform an operation. Creating a Method

In general, a method has the following syntax:

Modifier, returnvaluetype, methodname (list of parameters)

{

//Method Body;

}

A method definition consists of a method header and a method body like Modifiers, Return Type, and Method Name, Parameters, Method Body.Creating Objects,

Further we have learned that in general there are two different methods of passing an argument to a function. These methods are i) Call by value, ii. Call by reference. Call by value: In call by value method, the value of an argument is copied to the formal parameter. That is, the changes made in the actual argument are not reflected into the formal parameter, and Call by Reference, in Call by Reference method, the changes made to the actual argument are also reflected in the formal argument.

It is also understood that the constructors are not called in the main ( ) function. They are automatically executed at the time of object creation. For Example: the above program using default constructors can be written as DEFAULT CONSTRUCTOR. The default constructors are those constructors which do not accept any parameters/arguments. These constructors are automatically executed at the time of object creation. Another thing is related to Parameterised Constructors. The parameterised constructors are those constructors which accept parameters/arguments. At the time of object creation the values of these arguments/parameters are passed from main ( ) function.

There is learning about the ‘this’ keyword is used when a function will need to refer to the object which invoked it. The ‘this’ keyword can be used inside any method to refer to the current object. At the end of this Unit we came to know that the use of this keyword is redundant but perfectly correct. Inside rectangle ( ), this will always refer to the invoking object. When a local variable has the same name as an instance variable, the local variable hides the instance variable. That is why length and width were not used as the name of the parameters to the rectangle ( ) constructor inside the rectangle class. If we would have used in that way, then breadth would have referred to the formal parameter hiding the instance variable breadth

1.8 Suggested Answer for Check Your Progress

Check your progress 1

Answers: See Section 1.2

Check your progress 2

Answers: See Section 1.3

Check your progress 3

Answers: See Section 1.4

Check your progress 4

Answers: See Section 1.5

Check your progress 5

Answers: See Section 1.6

1.9 Glossary

1. **Modifiers -** is one which communicates to the compiler how to call the method. This defines the access type of the method.
2. **Return Type -** A method may return a value. The return Value Type is the keyword void.
3. **Method Name -** This is the actual name of the method. The method name and the parameter list together constitute the method signature.
4. **Parameters -** A parameter is like a placeholder. When a method is invoked, you pass a value to the parameter

1.10 Assignment

1. Write the steps to create objects.
2. Explain the process of creating a method.

1.11 Activities

Define a class student with data members studied, name, address, age and marks of 3 subjects. Provide the necessary constructors and methods along with getmarks() method and display the total and percentage marks with complete information about student.

1.12 Case Study

Define a class Rectangle with its length and breadth. Provide appropriate constructor(s), which gives facility of constructing Rectangle object with default values of length and breadth as 0 or passing value of length and breadth externally to constructor. Provide methods to calculate area and method display to display all information of Rectangle. Design different class Test Rectangle class in separate source file, which will contain the main function. From this main function, create an object which is a Rectangle and call the methods area and display.

1.13 Further Reading

1. Core Java 2, 2 volumes, Cay S. Horstmann, Gary Cornell, The Sun Microsystems Press, 1999, Indian reprint 2000
2. Java 2, The Complete Reference, Patrick Naughton and Herbert Schildt, Tata McGraw Hill, 1999
3. Programming with Java, Ed. 2,E. Balagurusamy, Tata McGraw Hill, 1998, reprint, 2000
4. The Java Tutorial, Ed. 2, 2 volumes, Mary Campione and Kathy Walrath, Addison Wesley Longmans, 1998
5. The Java Language Specification, Ed. 2, James Gosling, Bill Joy, Guy Steele & Gilad Bracha, (Ed. 1 1996, Ed. 2 2000), Sun Microsystems, 2000
6. Using Java 2, Joseph L. Weber, Prentice Hall, Eastern Economy Edition, 2000