UNIT 1 CLASS AND OBJECTS

Structure **Page Nos.**

1.0	Introduction	5
1.1	Objectives	5
1.2	Class Fundamentals	6
	1.2.1 Creating objects	
	1.2.2 Assigning object reference variables	
1.3	Introducing Methods	10
	1.3.1 Static methods	
	1.3.2 Constructors	
	1.3.3 Overloading constructors	
1.4	this Keyword	16
1.5	Using Objects as Parameters	17
	1.5.1 Argument passing	
	1.5.2 Returning objects	
1.6	Method Overloading	20
1.7	Garbage Collection	20
1.8	The Finalize ( ) Method	21
1.9	Summary	23
1.10	Solutions/Answers	23

1.0 INTRODUCTION

Object orientation is based on real life philosophy. In Unit 1 and 2 of Block 1 you are introduced to basic concepts of Object Oriented technology. It is emphasized that "in Object Oriented Programming (OOP) more emphasis is given on data" .To achieve this objective concept of classification is used which results into classes, that are the major component of object-oriented programming. Object Oriented Programming forces to think in terms of objects and interaction between objects them.

In this unit you will study how Java supports class, objects, and how objects are used in problem solving. You will also study how to use methods for communication between objects. We will use a special kind of methods known as static method in programs writing .To initialize objects at the time of creation we use constructors. Also in this unit we will learn the concept of constructor overloading. Further you will study use of ***this*** key word in programs. We will discuss the memory de–allocation technique used in Java.

Finally we will discuss various arguments (parameters) passing schemes, how an object is passed as parameter and object return from a function.

1.1 OBJECTIVES

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

- what is a class and how it is created in Java;
- what are objects and how they are declared;
- what are methods and their uses;
- what are static members;
- how to use constructors;
- how to use different argument passing techniques and use of objects as arguments in functions;
- how to return objects from function, and
- garbage collection and finalized method in Java.

1.2 CLASS FUNDAMENTALS

You have studied class in Block 1 of this course. Now you will learn how classes are defined and used in Java.

The class construct is what makes Java an object-oriented language. A Java class is a group of values with a set of operations to manipulate this values. Classes facilitate modularity and information hiding. Classes are used to define a new data type. Once a new data type is defined, variables of this data type can be created in program for solving problems. Variables of a class are known as objects of that class, and carry the properties of the class with values. Thus it can be said that, "a class is a template for an object of the properties and object is an instance of a class".

Before defining a class it must be clear to you for what purpose you are going to create class. i.e. " the nature and exact form of the class" should be reflected in class definition.

Now let us see how a class is defined in Java
General form of a Class:

```
class  className
{
type inst_var1;
type inst_var2;
type inst_var3;
…
…
type inst_varN;
type method_name1(arguments)
{
  // body of method
}
type methodname2 (arguments)
{
//body of method
}
…
….
}
```

You can see that class is declared by the use of the **class** keyword. Data and methods are defined within the class. Being the part of the class, data (variables) and methods are called member of the class. Data are called **member data** or **instance variable** of the class and methods are called **member functions** of the class. Member functions are defined within a class and act on the data member of the class.

As we have discussed before defining a class, the data members and member functions of a class should be decided. If we take Student class and want to display basic information of the student, first we have to decide the data members required to represent basic information, then we need a member function to display basic information.

Now you can see how a class Student is defined

```
class  Student

{
String name;
String  course
int  roll_no;
```

}

The class Student is not yet complete because in this class only data members are there. Still a member function to display basic information is needed.

Complete definition of class Student will have three data members-name, course and age and one member function display_info( ).

```
//class definition
class Student
{
String name;
String course ;
int age;
void display_info( ) // function for displaying basic information
{
System.out.println(" Student Information");
System.out.println("Name:"+name);
System.out.println("Course:"+course);
System.out.println("Age:"+age);
}
}
// end of Student class
```

As mentioned earlier class defines a new data type. In our case Student is a new data type. Now Student can be used to declare objects of Student class.
An object of Student class can be created as follows:

Student student_1 = new Student( );

As this statement is executed an object student 1 of Student class is created. You will see a detailed discussion of this type of statements in a later section of this unit.
Each time you create an object of a class a copy of each instance variables defined in the class is created. In other words you can say that each object of a class has its own copy of data members defined in the class. Member functions have only one copy and shared by all the objects of that class. All the objects may have their own value of instance variables. As given in *Figure1* every object of class Student will have its own copy of name, course, and age but only one copy of method display_info( ) for all the objects.

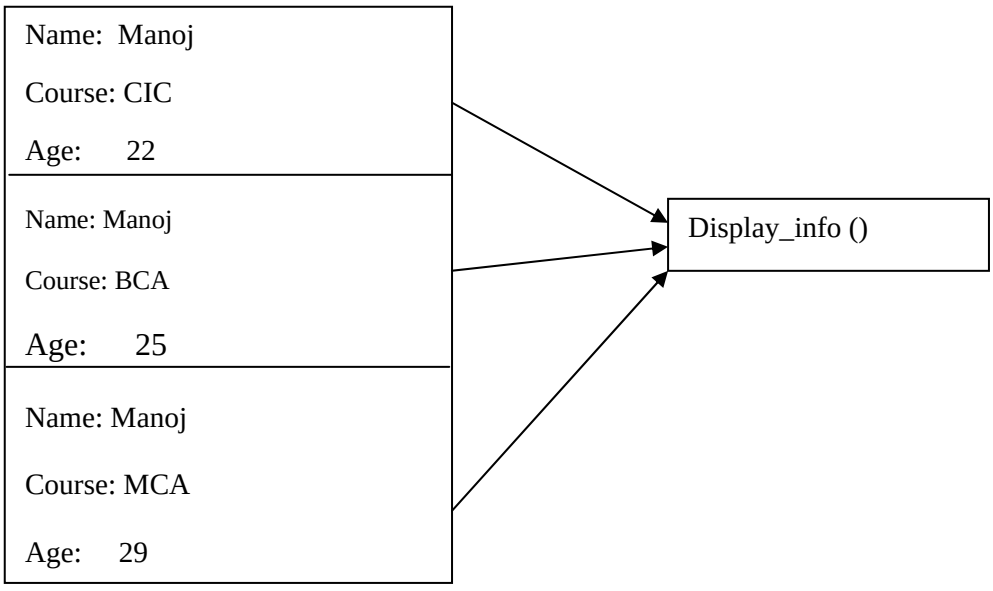

Figure 1: Objects of class Student

Now let us see how objects are declared and used inside a program.

1.2.1 Creating Objects

An object of a class can be created by performing these two steps.

1. Declare a object of class.
2. Acquire space for and bind it to object.

Why are two steps needed?

In the first step only a reference to an variable is created, no actual object's exists. For actual existence of an object memory is needed. That is acquired by using **new** operator.

The **new** operator in Java dynamically allocates memory for an object returns a reference to object and binds the reference with the object. Here reference to object means address of object. So it is essential in Java that all the objects must be dynamically allocated.

Declaring Student object

Step 1:
Student student1; // declaring reference to object

Step 2:
Student1 = new Student( ); // allocating memory

In normal practice these two steps are written in a single statement as
Student student1 = new Student( );

Now you can see a complete Java program for displaying basic information of students

```
//program
class Student
{
String name;
String course ;
int age;
void display_info( ) // function for displaying basic information
{
System.out.println(" Student Information");
System.out.println("Name:"+name);
System.out.println("Course:"+course);
System.out.println("Age:"+age);
}
}
// end of Student class
  class  Display_Test
{
public static void main( String para[])
{
Student  student1;
student1 = new Student();
sudent1.name = "Mr.Ravi"; //assigning value to name variable of  student1 object
student1.course = "MCA"; //assigning value to course variable of  student1 object
student1.age = 23; //assigning value to age variable of  student1 object
student1.display_info(); // invoking display_info( ) method on student1 object
}
}
```

The output of this program is:

```
Name:Mr.Ravi
Course:MCA
Age:23
```

If you observe this program, instance variables of student1 are assigned values. Is there any other way of assigning value to instance variables? For the questions the answer is yes. An object created can be used only if its instance variables contain value. The values of instance variables, represent the state of an objects for example, student1 object in this program. At present student is representing a student named Ravi, a MCA student of age 23.If you change the value of any of the instance variable's the state of the object will change.

Two approaches can be used for assigning value to the instance variables of the objects. In first approval create an object and initialize instance variables one by one, as done in the above program. Initialization of instance variable is done by using dot(.) operator. Dot(.) operator is used to access members (data and method both) of the objects.

object_name.variable_name = value;

But this method of initialization of objects is not convenient because all instance variables of the object must be initialized carefully, and this exercise should be done for all objects before they are used. In this method of initialization there is a chance of skipping some variables unassigned where large number of objects are to be initialized.

To overcome this problem second approach of object initialization with the help of **constructors,** can be used. We will discuss how constructors are used in the next section of this unit.

1.2.2 Assigning Object Reference Variables

One object can be assigned to another object, of same type but it works very different than a normal assignment. Let us see it with the help of a program.

```
class Person
{
String name;
int age ;
String address;
void Display( )
{
System.out.println("Person Information:"+name +" ("+age +")"+"\n"+address);
}
}
class Reference_Test
{
public static void main(String[] args)
{
Person p = new Person();
Person q = new Person();
p.name= "Mr.Naveen Kumar";
p.age= 24;
p.address = "248,Sector 22, Noida";
p.Display();
q = p;// q refer to p
q.name = "Mr.Suresh";
q.address = "22,Mahanadi,IGNOU,Maidan Garhi";
p.Display();
```

```
q.Display();
}
}
```

Output of this program is:
Person Information:Mr.Naveen Kumar (24)
248,Sector 22, Noida
Person Information:Mr.Suresh (24)
22,Mahanadi,IGNOU,Maidan Garhi
Person Information:Mr.Suresh (24)
22,Mahanadi,IGNOU,Maidan Garhi

In this program two objects p and q are created. Object p is initialized with some values, then Display method is called through object p. It displays the information of object p. Further, object p is assigned to object q as reference variable. You can see in the figure both object p and object q are referring to the same object. Thus any change made in object p or q changes will be reflect the both p and q. You can see in the program that changes made in name and address by q are reflected in p also. Whenever in a program this type of referencing of variables is used care should be taken while changing the values of instance variables of object being referred.

An object is a class instance. The class of an object determines what it is and how it can be manipulated. A class encapsulates methods, data, and implementation of methods. This encapsulation is like a contract between the implementer of the class and the user of that class.

Check Your Progress 1

1) Explain the process of object definition in Java.

……

……

2) What is the advantage of having member data and member functions within the class?

……

……

3) When an object be used as reference of another object? What care should be taken in such kind of referencing?

……

……

1.3 INTRODUCING METHODS

A Java class is a group of values with a set of operations. The user of a class manipulates object of that class only through the methods of that class.

General form of a method is:

```
type name_of_method (argument_list)
{
// body of method
```

```
}
```

type specifies the type of data that will be returned by the method. This can be any data type including class types. In case a method does not return any value, its return type must be void. The argument–list is a list of type and identifier pairs separated by commas. Arguments are basically variables that receive the values at the time of method invocation. If a method has return type other than void, it returns a value to the calling point using the following form of statement.

```
return value;// value is the value to be returned by function.
```

Suppose we define a class to represent complex numbers. The complex class definition shown in the program given below illustrates how this can be done. Two variables real and imag, are declared. These represent the real and imaginary parts of a complex number (respectively). The program also defines three methods, assignReal and assign Imag() and showComplexl()that can be used to assign values to the real is () part imaginary part of a complex number, and to show the real number respectively.

```
class Complex
{
double real;
double imag;
void assignReal( double r)
{
real = r;
}
void assignImag( double i)
{
imag= i;
}
void showComplex ( )
{
System.out.println("The Complex Number is :"+ real +"+i"+imag);
}
}
class Complex_Test
{
public static void main(String[] args)
{
Complex R1 = new Complex();
R1.assignReal(5);
R1.assignImag(2);
R1.showComplex();
}
}
```

Output of this program is:

```
The Complex Number is :5.0+i2.0
```

Sometimes we need to have the same name of different methods in a class. All these methods perform different operations of a similar nature. For example of two add methods in a class one is used for adding two integer variables and the other for adding two double variables. Methods of the same name in a class are called overloaded methods, and the process of defining this type of methods is called method overloading. The concept of overloading we cover in more detail in the next section of this unit. The concept of overloading can also be used in case of defining constructors. Constructor overloading we will discuss in section 1.3.3 this unit.

So far we have seen that the initialization of instance variables is done after creating objects. Now we will use constructors for initialization of instance variables.

1.3.1 Static Methods

Methods and variables that are not declared as *static* are known as instance methods and instance variables or in other words you can say that they belong to objects of the

. To refer to instance methods and variables, you must instantiate the class first, then obtain the methods and variables from the instance.

A static method is a characteristic of a class, not of the objects it has created.

Static variables and methods are also known as class variables or class methods since each class variable and each class method occurs once per class. Instance methods and variables occur once per instance of a class or you can say every object is having its own copy of instance variables.

One very important point to note here is that a program can execute a static method without creating an object of the class. All other methods must be invoked through an object, and, therefore an object must exist before they can be used. You have seen that every Java application program has one main()method. This method is always static because Java starts execution from this main()method, and at that time no object is created.

Let us see one example program:

```
import Java.util.Date;
class DateApp
{
public static void main(String args[])
{
Date today = new Date();
System.out.println(today);
}
}
```

The last line of the main() method uses the **System** class from the Java.lang package to display the current date and time. See the line of code that invokes the println () method.

System.out.println(today);

Now look at the details of the argument passed to it.

System.out refers to the out variable of the System class. You already know that, to refer to static variables and methods of a class, you use a syntax similar to the C and C++ syntax for obtaining the elements in a structure. You join the class's name and the name of the static method or static variable together with a dot (.)

The point which should be noticed here is that the application never instantiated the System class and that out is referred to directly from the class. This is because out is declared as *a* static variable: *a* variable associated with the class rather than with an instance of the class. You can also associate methods with a class--*static methods*--using static keyword.

1.3.2 Constructors

A constructor initializes object with its creation. It has the same name as the name of its class. Once a constructor is defined, it is automatically called immediately the memory is allocated before the **new** operation completes. Constructor does not have any return type, it implicit return type is class object. You can see constructor as a class type. Its job is to initialize the instance variables of an object, so that the created object is usable just after creation.

In program Complex_Test you have seen that for initializing imaginary and real parts, two methods have been used. Both the methods have been invoked on the object one by one. In place of methods if you use constructor for initialization you have to make

changes in Complex_Test program. Remove methods used for initializing variables and use one method having same name of the class, i.e., Complex with two arguments.

```
class Complex
{
double real;
double imag;
Complex( double p, double q)
{
System.out.println("Constructor in process...");
real = p;
real = p;
imag = q;
}
void showComplex ( )
{
System.out.println("The Complex Number is :"+ real +"+i"+imag);
}
}
class Complex_Test
{
public static void main(String[] args)
{
Complex R1 = new Complex(5,2);
R1.showComplex();
}
}
```

The output of this program is:

```
Constructor in process…
The Complex Number is :5.0+i2.0
```

If you compare this program and the previous Complex_Test program you will find that in this program the instance variable of object R1 is initialized with the help of constructor Complex (5, 2), value 5 has been assigned to the real variable and 2 has been assigned to the imag variable. In the previous program, to initialize these variables, methods assignReal() and assignImag() were used.

Constructors can be defined in two ways. First , a constructor may not have parameter. This type of constructor is known as non-parameterized constructor. The second, a constructor may take parameters. This type of constructor is known as parameterized constructor.

If non-parameterized constructor is used for object creation, instance variables of the object are initialized by fixed values at the time of definition of constructor itself.

You can see this program

```
class Point
{
int x;
int y;
Point()
{
x= 2;
y= 4;
}
void Display_Point()
{
System.out.println("The Point is: ("+x+","+y+")");
}
```

```
}
class Point_Test
{
public static void main( String args[])
{
Point p1 = new Point();
Point p2 = new Point();
p1.Display_Point();
p2.Display_Point();
}
}
```

Output of this program is:
The Point is: (2,4)
The Point is: (2,4)

Constructor Point is a non-parameterized constructor. Both the objects p1 and p2 are created by using Point constructor and are initialized by the same value which is assigned during definition of the constructor. This type of constructors is generally used for the initialization of those objects for which the initial value of instance variables is known. If values of the instance variables are given at the time of object creation, parameterized constructors are used for this.

For example, if you want to create Point class object with your initial values of x and y coordinates, you can use parameterized constructor. You have to make the following changes in the previous program.

1. In place of non-parameterized constructor, define parameterized constructor.
2. Pass appropriate values as arguments to constructors.

```
class Point
{
int x;
int y;
Point(int a, int b)
{
x= a;
y= b;
}
void Display_Point()
{
System.out.println("The Point is: ("+x+","+y+")");
}
}
class Point_1_Test
{
public static void main( String args[])
{
Point p1 = new Point(2,5);
Point p2 = new Point(9,7);
p1.Display_Point();
p2.Display_Point();
}
}
```

Output of this program is:
The Point is: (2,5)
The Point is: (9,7)

In this program you can see that points p1 and p2 are initialized by values of programmer's choice by using parameterized constructor.

1.3.3 Overloading Constructors

You may ask whether there can be more than one constructor in a class. Yes, there may be more than one constructor in a class but all the constructors in a class either will have different types of arguments or different number of arguments passed to it. This is essential because without this it wouldn't be possible to identify which of the constructors is invoked. Having more than one constructor in a single class is known as constructor overloading. In the program given below more than one constructor in class Student are defined.

```
class Student
{
String name;
int roll_no;
String course;
Student( String n, int r, String c)
{
name = n;
roll_no  = r;
course = c;
}
Student(String n, int r )
{
name  = n;
roll_no = r;
course = "MCA";
}
void Student_Info( )
{
System.out.println("********** Student Information ***********");
System.out.println("Name:"+name);
System.out.println("Course:"+course);
System.out.println("Roll Number:"+roll_no);
}
}
class Student_Test
{
public static void main(String[] args)
{
Student s1 = new Student("Ravi Prakash", 987770012,"BCA");
Student s2 = new Student("Rajeev ", 980070042);
s1.Student_Info();
s2.Student_Info();
}
}
```

Output of this program is:

```
********** Student Information ***********
Name:Ravi Prakash
Course:BCA
Roll Number:987770012
********** Student Information ***********
Name:Rajeev
Course:MCA
Roll Number:980070042
```

Both the constructors of this program can be used for creating two different types of objects. Here different types are simply related to the values assigned to instance variables of objects during their initialization through constructors. By using a constructor with three arguments all the three instance variables name, roll_no and, course can be initialized. If a constructor with two arguments is used to create object, only instance variables name and roll_no can be given initial value through constructor and the variable course is initialized by a constant value "MCA".

Check Your Progress 2

1) What is the need of member function in a class? Explain through a program how member functions are defined in Java.

……

……

2) Explain why the main method in Java is always static?

……

……

3) Explain the use of constructor with the help of a program.

……

……

4) Write a program in Java to create Bank_Account class, which defines two different constructors to create objects.

……

……

1.4 this KEYWORD

If a method wants to refer the object through which it is invoked, it can refer to by using **this** keyword. You know it is illegal to have two variables of the same name within the same scope in a program. Suppose local variables in a method, or formal parameters of the method, overlap with the name of instance variables of the class, to differentiate between local variables or formal parameters and instance variables, **this** keyword is used. The reason to use **this** keyword to resolve any name conflict that might occur between instance variables and local variable is that this keyword can be used to refer to the objects directly. You can see in this program class Test_This is having one variable named *rate* and method. Total_Interest is also having one local variable named *rate*. To avoid conflict between both the rate variables keyword has been used with *rate* variable of Test_This class.

```
class  Test_This
{
int rate ;
int amount;
int interest;
Test_This( int r, int a)
{
rate = r;
amount =a;
}
void Total_Interest( )
```

```
{
int rate = 5;
rate = this.rate+rate;
interest = rate*amount/100;
System.out.println("Total Interest on "+amount+" is: "+interest);
}
}
class  This_Test
{
public static void main(String[] args)
{
Test_This Ob1 = new This( 5, 5000);
Ob1.Total_Interest();
}
}
```

Output of this program is:
Total Interest on 5000 is: 500.

1.5 USING OBJECTS AS PARAMETERS

Objects can be based as parameter and can also be returned from the methods. Let us look at the two aspects one by one.

1.5.1 Argument Passing

During problem solving you need to pass arguments through methods. Basically by passing arguments you can generalize a method. During problem solving generalized methods can be used for performing operations on a variety of data.

You can see in this program below that for find the area of square and rectangle. The methods Area_S and Area_R for finding area of square and rectangle respectively are defined.

```
class Test
{
int Area_S( int i)
{
 return i*i;
}
int Area_R(int a,int b)
{
 return a*b;
}
}
class Area_Test
{
public static void main(String args[])
{
 Test t = new Test();
 int area;
 area = t.Area_S(5);
 System.out.println("Area of Square is : "+area);
 area = t.Area_R(5,4);
 System.out.println("Area of Rectangle is : "+area);
}
}
```

Output of this program is:
Area of Square is: 25

Area of Rectangle is: 20

Can you tell the way of passing parameters in methods Area_S and Area_R? It is call by value. Right as you have studied about two basic way pass by value and pass by reference of parameter passing in functions in course: MCS 01.

Now you may ask a question whether parameter passing in Java is by reference or by value. The answer is everything in Java except value, passed by reference.

Pass-by-value means that when you call a method, a copy of the *value* of each of the actual parameter is passed to the method. You can change that copy inside the method, but this will have no effect on the actual parameters. You can see in the program given below. The method max has two parameters that are passed by value.

```
class Para_Test
{
static     int  max(int a, int b)
{
if (a > b)
return a;
else
return  b;
}
public static void main(String[] args)
{
int num1 = 40, num2 = 50, num3;
num3 = max( num1, num2);
System.out.println("The maximum in "+num1 +" and "+num2+" is : "+num3);
}
}
```

Output of this program is:

The maximum in 40 and 50 is: 50

The values of variables are always primitives or references, never objects. In Java, we can pass a reference of the object to the formal parameter. If any changes to the local object that take place inside the method will modify the object that was passed to it as argument. Due to this reason objects passing as parameter in Java are referred to as passing by reference. In the program given below object of the class Marks is passed to method Set_Grade, in which instance variable grade of the object passed is assigned some value. This reflects that the object itself is modified.

```
class Marks
{
String name;
int percentage;
String grade;
Marks(String n, int m)
{
name = n;
percentage = m;
}
void Display()
{
System.out.println("Student Name :"+name);
System.out.println("Percentage Marks:"+percentage);
System.out.println("Grade : "+grade);
System.out.println("*****************************");
}
}
class Object_Pass
{
public static void main(String[] args)
```

```
{
Marks ob1 = new Marks("Naveen",75);
Marks ob2 = new Marks("Neeraj",45);
Set_Grade(ob1);
System.out.println("*****************************");
ob1.Display();
Set_Grade(ob2);
ob2.Display();
}
static void Set_Grade(Marks m)
{
if (m.percentage >= 60)
m.grade ="A";
else if( m.percentage >=40)
m.grade = "B";
else
m.grade = "F";
}
}
```

Output of this program is:

```
*****************************
Student Name :Naveen
Percentage Marks:75
Grade : A
*****************************
Student Name :Neeraj
Percentage Marks:45
Grade : B
*****************************
```

1.5.2 Returning Objects

Like other basic data types, methods can return data of class type, i.e. object of class. An object returned from a method can be stored in any other object of that class like as values of basic type returned are stored in variables of that data type. You can see in the program below where an the object of class Salary is returned by method the incr_Salary.

```
class Salary.
{
int basic ;
String E_id;
Salary( String a, int b)
{
E_id = a;
basic = b;
}
Salary  incr_Salary ( Salary s )
{
s.basic = basic*110/100;
return s;
}
}
class Ob_return_Test
{
public static void main(String[] args)
{
Salary s1 = new Salary("I100",5000);
Salary s2;// A new salary object
s1 = s1.incr_Salary( s1);
```

```
System.out.println("Current Basic  Salary is : "+ s2.basic);
}
}
```

Output of this program is:
Current Basic Salary is: 5500

Now once again you can return to a discussion of overloading, which was left at section 1.3 of this unit.

1.6 METHOD OVERLOADING

Sometimes two or more methods which are very similar in nature are required. For example, you can take methods Area_S and Area_R in a class Test in program of section 1.3 of this unit. Both the methods are finding area but the argument passed to them and their implementation is different. All the methods of similar kinds in a class can have the same name, but remember all the methods will have different prototypes. This concept is known as method overloading. By matching the type and order of arguments passed to methods. The exact method that should execute is resolved. Sometimes overloaded methods may have different return types, the return type alone is not sufficient to differentiate two overloaded methods. You can see in the program given below that the area()method is overloaded.

```
class Test
{
int Area( int i)
{
return i*i;
}
int Area(int a,int b)
{
return a*b;
}
}
class Area_Overload
{
public static void main(String args[])
{
Test t = new Test();
int area;
area = t.Area(5);
System.out.println("Area of Squire is : "+area);
area = t.Area(5,4);
System.out.println("Area of Rectangle is : "+area);
}
}
```

Output of this program is:
Area of Squire is: 25
Area of Rectangle is: 20.

1.7 GARBAGE COLLECTION

One of the key features of Java is its garbage-collected heap, which takes care of freeing dynamically allocated memory that is no longer referenced. It shields the substantial complexity of memory allocation and garbage collection from the developer. Because the heap is garbage-collected, Java programmers don't have to explicitly free the allocated memory.

The JVM's heap stores all the objects created by an executing Java program. You know Java's "new" operator allocate memory to object at the time of creation on the heap at run time. Garbage collection is the process of automatically freeing objects that are no longer referenced by the program. This frees the programmer from having to keep track of when to free allocated memory, thereby preventing many potential bugs and thus reduces the programmer's botheration.

The name "garbage collection" implies that objects that are no longer needed by the program are "garbage" and can be thrown away and collects back into the heap. You can see this process as "memory recycling". When the program no longer references an object, the heap space it occupies must be recycled so that the space is available for subsequent new objects. The garbage collector must somehow determine which objects are no longer referenced by the program and make available the heap space occupied by such unreferenced objects.

Advantages and Disadvantages of garbage collection

Giving the job of garbage collection to the JVM has several advantages. First, it can make programmers more productive. Programming in non-garbage-collected languages, the programmer has to spend a lot of time in keeping track of the memory de-allocation problem. In Java, the programmer can use that time more advantageously in doing other work.

A second advantage of garbage collection is that it ensure program integrity. Garbage collection is an important part of Java's security strategy. Java programmers feel free from the fear of accidental crash of the system because JVM is there to take care of memory allocation and de–allocation.

The major disadvantage of a garbage-collected heap is that it adds an overhead that can adversely affect program performance. The JVM has to keep track of objects that are being referenced by the executing program, and free unreferenced objects on the fly. This activity will likely take more CPU time than would have been required if the program explicitly freed unrefined memory. The second disadvantage is the programmers in a garbage-collected environment have less control over the scheduling of CPU time devoted to freeing objects that are no longer needed.

1.8 THE FINALIZE () METHOD

Sometimes it is required to take independent resources from some object before the garbage collector takes the object. To perform this operation, a method named finalized () is used. Finalized () is called by the garbage collector when it determines no more references to the object exist.

Finalized()method has the following properties:

1. Every class inherits the **finalize()** method from **Java.lang.Object**.

2. The garbage collector calls this method when it determines no more references to the object exist.

3. The **Object class** finalize method performs no actions but it may be overridden by any derived class.

4. Normally it should be overridden to clean-up **non-Java** resources, i.e. closing a file, taking file handle, etc.

Writing a finalize() Method

Before an object is garbage collected, the runtime system calls its finalize() method. The intent for this to release system resources such as open files or open sockets before object getting collected by garbage collector.

Your class can provide for its finalization simply by defining and implementing a finalize() method in your class. Your finalize() method must be declared as follows:

protected void finalize () throws throwable

This class opens a file when its constructed:

```
class OpenAFile
{
FileInputStream aFile = null;
OpenAFile(String filename)
{
try
{
a File = new FileInputStream(filename);
}
catch (Java.io.FileNotFoundException e)
{
System.err.println("Could not open file " + filename);
}
}
}
```

To avoid accidental modification or other related problem the OpenAFile class should close the file when it is finalized. Implementation of finalize () method for the OpenAFile class:

```
protected void finalize () throws throwable
{
if (aFile != null)
{
a File.close();
a File = null;
}
}
```

The Java programmer must keep in mind that it is the garbage collector that runs finalizers on objects. Because it is not generally possible to predict exactly when unreferenced objects will be garbage collected, and to predict when object finalizers will be run. Java programmers, therefore, should avoid writing code for which program correctness depends upon the timely finalization of objects. For example, a finalize of an unreferenced object may release a resource that is needed again later by the program. The resource will not be made available until after the garbage collector has run the object finalizer. If the program needs the resource before the garbage collector has run the finalizer, the program is out of luck.

Check Your Progress 3

1) What is the need of parameter passing? Show through a program how parameters are passed to methods.

……

……

2) What are two major advantages of automatic de-allocation of memory?

……

……

3) Write a program in Java, in which a method named add is overloaded. The add method sums two integer values, one integer value and one double value, two double values.

……

……

1.9 SUMMARY

In this unit, the concept of class is discussed. The process of object creation using class is explained. Every object has its state and behavior. Objects behavior are defined inside the class in terms of member functions. In this unit are also discussed about creation of constructors and their advantages. The need of static methods is explained with the help of the main () method of Java application program, which is always static. This unit also explained how arguments are passed to a member function. In the last section of this unit "Garbage Collection" one of the very important concepts of Java programming is explained.

1.10 SOLUTIONS/ANSWERS

Check Your Progress 1

1) First define the class for the object to be declared. Then using new operator allocates needed space to the variables of object.

Object definition in Java is a two-step process

i Declare variable of class type.

ii. Using new operator allocate needed space to the object.

For example to define object of Book class do the following:
Book book1 // declaration of object named book1
book1 = new Book(); // allocating memory.

2) Objects used in problem solving are instances of classes. Objects are treated as black box with public interfaces. Interfaces are provided with the help of member functions.

Member functions and member data are kept inside the class to ensure that data don't get accessed and modified accidentally. Definition of member functions is inside the class and it provides scope of modification in the definition without affecting outside world.

3) One object can be used as reference to another object provided both are of the same class type. Objects should be used as a reference very carefully because if there is any change in values of instance variables of one object, values of instance variable of the second object also gets changed.

Check Your Progress 2

1) Member functions in a class are required for providing features of the objects.

In the program given below to display the price of book object member function Get_Price() is defined.

```
//program
class Book
{
String Title;
String Author;
int Price;
Book( String t, String a, int p)
{
Title = t;
Author = a;
 Price = p;
}
void Get_Price ( )
{
 System.out.println("Price of  the book is :"+Price);
}
}
class Book_Test
{
public static void main(String args[])
{
Book b1 = new Book("Java Programming","Dr. Rajkumar Singh",250);
b1.Get_Price();
}
}
```

Output:
Price of the book is: 250

2) A static method can be invoked through the class itself rather than through an object as in the case of member functions. There is no need of any object to call a static method. Any static method is part of class not the part of objects. When *main* method is called there does not exist any object. Therefore, it become essential to have main () method as *static*.

3) Constructors are used to initialize the objects. Objects created by using constructors are ready to use. The program given below creates objects of Complex class using constructor defined in this class.

```
//program
class Complex
{
 int real;
int imaginary;
Complex ( int r, int i)
{
 real = r;
imaginary  = i;
}
void ShowNumber()
{
System.out.println("The Number is :" + real+"+" +imaginary+"i");
}
}
public class Cons_Test
{
public static void main(String args[])
{
Complex c= new Complex(5,3);
```

```
c.ShowNumber();
}
}
```

Output
The Number is :5+3i

4) In this program there are two different constructors of Bank_Account class.

```
//program
class Bank_Account
{
 String Name;
 int  Account_No;
 String Address;
 int  Init_Bal;
Bank_Account(String n, int a)
{
Name =  n;
Account_No = a;
}
Bank_Account (String n, String addr, int a , int b)
{
Name =  n;
Address = addr;
Account_No = a;
Init_Bal = b;
}
}
public  class Account_Test
{
public static void main( String args[])
{
Bank_Account Ac1 = new Bank_Account("Mr. Naveen", 11002345);
Bank_Account Ac2 = new Bank_Account( "Mr. Suresh", "K-2 MGI Basant Kunj
New Delhi", 12001347, 500);
System.out.println("Account No. of  "+Ac1.Name + " is  :"+Ac1.Account_No);
System.out.println("Initial Balance in account of "+Ac2.Name+" is
Rs."+Ac2.Init_Bal);
}
}
```

Output
Account No. of Mr. Naveen is :11002345
Initial Balance in account of Mr. Suresh is Rs.500.

Check Your Progress 3

1) Whenever there is a need to pass some value to a method from outside the values are passed as arguments to method. Example' is a method written to calculate interest on some amount. Interest rate is a variable that can be passed as argument to the method used for interest calculation.

In the program given below Interst_Calc method is defined

```
//program
class Interest
{
int amount ;
Interest( int a)
{
amount = a;
```

```
}
int Interest_Calc (int rate)
{
return amount * rate / 100;
}
}
public class ParaTest
{
public static void main (String args[])
{
Interest  inter = new Interest(5000);
 System.out.println("The Inters amount for Rs 5000 at the rate of 10 % for a year is:"+
inter.Interest_Calc(10));
}
}
```

Output
The Interest amount for Rs 5000 at the rate of 10 % for a year is: 500

2) There are two major advantages of automatic garbage collection
   i. Increase programmer productivity.
   ii. Ensure program integrity.

3) //Program

```
class Add_Overload
{
Add_Overload()
{
 System.out.println("*******Welcome to Overload Demo********");
}
void add( int i, int j)
{
System.out.println("Inside add(int,int)and the Sum is :"+(i+j));
}
void add( int i, double j)
{
 System.out.println("Inside add(int,double)and the Sum is :"+(i+j));
}
void add( double i, double j)
{
System.out.println("Inside add(double,double)and the Sum is:"+(i+j));
}
}
public class OverloadTest
{
public static void main (String args[])
{
Add_Overload  Ol  = new  Add_Overload();
Ol.add(34,35);
Ol.add( 25, 47.67);
Ol.add(12.5,25.5);
}
}
```

Output:

```
*******Welcome to Overload Demo********
Inside add(int,int)and the Sum is:69
Inside add(int,double)and the Sum is:72.67
Inside add(double,double)and the Sum is:38.0
```