UNIT 4 CLASS AND OBJECTS

Structure **Page Nos.**

4.0 INTRODUCTION

In Unit 1 of this Block you have learned basic features of Object Oriented programming. As of now you are aware of the basic concepts of Object Orientation. When we look at object orientation closely, it is observed that the principles used in this approach are based on real-life philosophy. In Object Oriented Programming (OOP), data is given emphasis. To achieve OOP objectives, the concept of classification and encapsulation in designing and defining classes. As classes are the core component of object-oriented programming, you need to focus on identifying and defining them in your program to develop any solution. OOP makes you think about objects and interaction between objects while developing solutions.

In this unit, you will study how class and objects are defined in Java. You will learn to create and use objects in problem-solving. You will also study how to create methods and call them in your program. In this unit, you will learn to define the constructor which is used to initialize the objects. Further, you will learn the use of ***this*** keyword in writing programs. Also, this unit explains the memory deallocation and garbage collection in Java and the use of finalize method.

4.1 OBJECTIVES

After going through this unit you will be able to:

- Define class and objects in Java,
- Define and use methods in Java program,
- Use the concept of Encapsulation in Java programming,
- Use Access Modifiers,
- Define Constructors in Java programs,

- Use this keyword ,
- Pass objects as parameters,
- Describe garbage collection in Java , and
- Use finalize ( ) method in Java Program.

4.2 CLASS FUNDAMENTALS

You have been introduced the concept of class in unit 1 of this Block. Now let us learn how classes are defined and used in Java.

The class is basic construct which makes Java an object-oriented programming language. A class is a group of values with a set of operations to manipulate these values. The operation of the class provides functionality to it. The group of values are used to represent the attribute (characteristic) of the objects belonging to the class. Classes bring modularity to the system and also provide the scope of implementing information hiding. Classes are used for defining a new data type. Once you define a new data type, you can create variables of this data type and use them in your program. Classes are used to create *objects.* The *objects* carry the properties of the class with values and actually participate in problem solving. Thus it can be said that *"a class is a template for an object, and object is an instance of a class".* You can create as many instances of a class as required in your program for problem-solving.

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

Now let us see how a class is defined in Java.

The general form of a Class:

```
class  NameOfClass
{
type var1;
type var2;
…
type varN;
return type Method1( list of arguments….)
{
  // body of method
}
return type Method2 ( list of arguments…….)
{
//body of method
}
}
```

A class is declared using the ***class*** *keyword*. Inside the class, variables and methods are defined. The data (variables) and methods of the class are called a *member* of the class. Data are called ***member data*** *or* ***instance variables*** of the class, and methods are called ***member functions*** of the class. The member functions are defined inside the body of the class and perform the desired objective for which they are implemented. Ome method should fulfil only one well defined objective.

Before you define a class, it is necessary that you decide what should be the data members and member functions of that class. For that, you as a programmer need to know the basic requirements that should be fulfilled by the class you are going to crate. If we take the example of Student class and want to display the student's basic information, first you should decide the data members required for representing students' basic information, and then you need to identify the member functions that will serve the basic functionality. A member function that may display the basic information is identified and implemented in the example below.

Now you can define a class Student as:

```
 class  Student
{
String name;
String  course
int  roll_no;
}
```

The class Student declared above is not yet complete because only data members are declared in this class. Still as per our need a member function to display basic information is to be defined. So, complete class student definition will have three data members : name, course and age, and one member function Display_Info( ).

```
//class definition
class Student
{
String name;
String course ;
int roll_no;
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(“Roll Number:”+roll_no);
System.out.println(“Age:”+age);
}
}
// end of Student class
```

As mentioned earlier, a class defines a new data type. In this case, now Student is a new data type, and Student can be used to declare objects of this class.

An object of Student class can be created and memory assigned as follows, using *new* operator:

Student student1 = new Student( );

When the statement given above is executed, an object named *student1* of type Student is created. You will see a detailed discussion of this type of statement in a later section of this unit. Once an object is created, it can be used for performing the defined operations. Each time you create an object of a class, a copy of each instance variable 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 is shared by all the objects of that class.   All the objects may have their own value of instance variables. As shown in *Figure 4.1, there are three objects* of the Student class, and every object has its own copy of name, course, and age, but only one copy of method Display_Info( ) for all the objects.

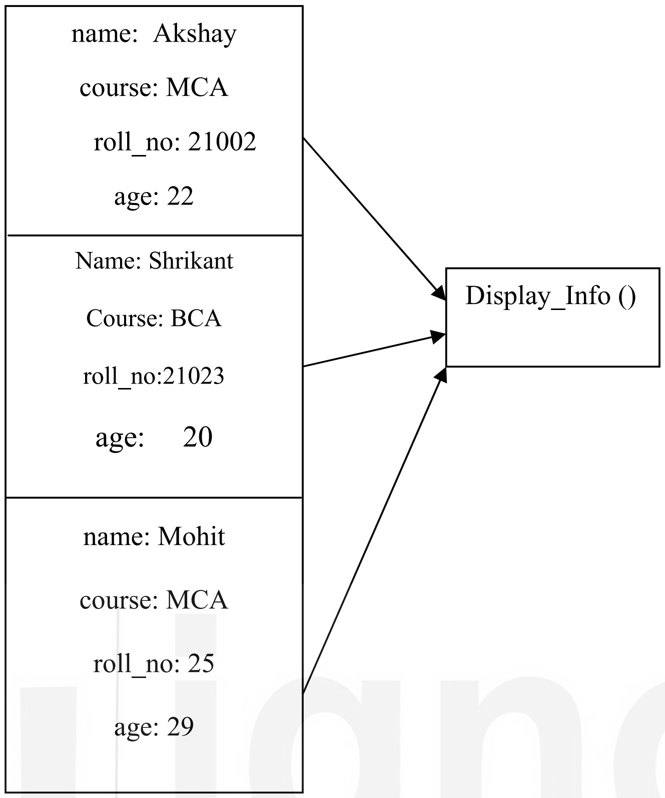


Figure 4.1: Objects of Student class

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 an object of the class, similar to the declaration of any other variables you have done so far.
2. Acquire space for an object and bind it to the object.

Can you guess why two steps are used?

The first point indicates that first, only a reference to a variable(object) is created And at this stage, no actual object exists. The memory needs to be allocated for the actual existence of an object, and that is done using the **new** operator.

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

Declaring objects of Student class

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( );

ignou THE PEOPLE'S UNIVERSITY

There are two ways to assign value to the object's instance variables. The first way is to create an object and initialize all the instance variables of the objects one by one, as shown in the Java program given below. Initialization of instance variable is done by using dot(.) operator. Remember that the dot(.) operator is used to access the object's members (data and method both).

object_name.variable_name = value;

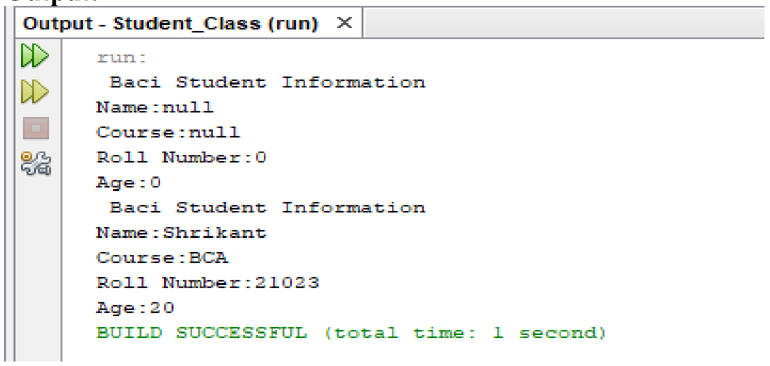
The way mentioned above for initialising objects is not convenient as you have to carefully initialize all instance variables of the object. This exercise should be performed for all objects before they are used in any operation of the program. In this way of initialization, when there is large numbers of objects to be initialized, there is a chance that the programmer may skip some variables unassigned. To overcome this problem second approach of object initialization may be performed using **constructors.** We will discuss how constructors are used in the coming section of this unit.

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

Java Program

```
package student_class;
class Student
{
String name;
String course ;
int roll_no;
int age;
void Display_Info( ) // method to displaying basic information
{
System.out.println(" Basic Student Information");
System.out.println("Name:"+name);
System.out.println("Course:"+course);
System.out.println("Roll Number:"+roll_no);
System.out.println("Age:"+age);
}
}
// end of Student class
public class Student_Class
{
public static void main( String para[])
{
Student  student1;//object declaration
student1 = new Student(); // Assigning memory
student1.Display_Info(); // invoking Display_Info( ) method  on an object which is not
initialised
student1.name = "Shrikant"; //assigning value to name variable of  student1 object
student1.course = "BCA"; //assigning value to course variable of  student1 object
student1.roll_no = 21023; //assigning value to roll_no variable of  student1 object
student1.age = 20; //assigning value to age variable of  student1 object
student1.Display_Info(); // invoking Display_Info( ) method on student1 object
}
}
```

Output:



```
run:
 Baci Student Information
Name:null
Course:null
Roll Number:0
Age:0
 Baci Student Information
Name:Shrikant
Course:BCA
Roll Number:21023
Age:20
BUILD SUCCESSFUL (total time: 1 second)
```

If you observe this program, instance variables of the object named student1 are assigned values. You may ask, is there any other way of assigning value to instance variables? For this question, the answer is yes, and you will learn about it later in this unit. An object is usable only if its instance variables contain some value. The values of instance variables represent the state of that objects; for example, the student1 object in the above program has assigned some value to its data members. The object student1 represents a student named Shrikant, a BCA student of age 20 whose Roll Number is 21023. When there is any change in the value of any of the instance variables, then the state of the object changes. Also, note that when Display_Info() method is invoked without initializing the member data, then default values of member data are displayed. It means if you do not initialize any member data, it will contain a default value or garbage value.

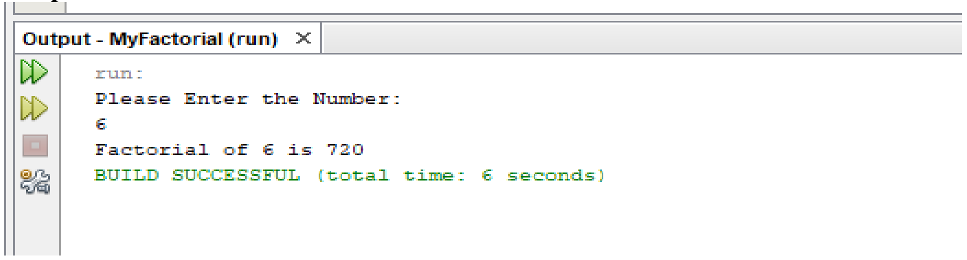
Also, an example program to find the factorial of a given number is written here. This program is just to show you how to create class and objects and use them. You already know the use of Scanner class and its methods in this program. Also you are aware of using Integer.parseInt(S) method which converts string to integer.

Java Program

```
package myfactorial;
import java.util.Scanner;  //Scanner class
// Java program to find factorial of  a given number
class Factorial
{
  //int number;
// method to find factorial of given number
   int factorial( int n)
   {
        if (n == 0)
        return 1;
        return n*factorial(n-1);
   }
}
public class MyFactorial
{
public static void main(String[] args)
   {
  Factorial Fact = new Factorial();
  //read the number
   Scanner myObj1 = new Scanner(System.in);  // Scanner object
   System.out.println("Please Enter the Number:");
   String S = myObj1.nextLine();
   int number = Integer.parseInt(S);
   System.out.println("Factorial of "+ number + " is " + Fact.factorial(number));
```

```
    }
}
```

Output:



Output - MyFactorial (run)

```
run:
Please Enter the Number:
6
Factorial of 6 is 720
BUILD SUCCESSFUL (total time: 6 seconds)
```

1.2.2 Assigning Object Reference Variables

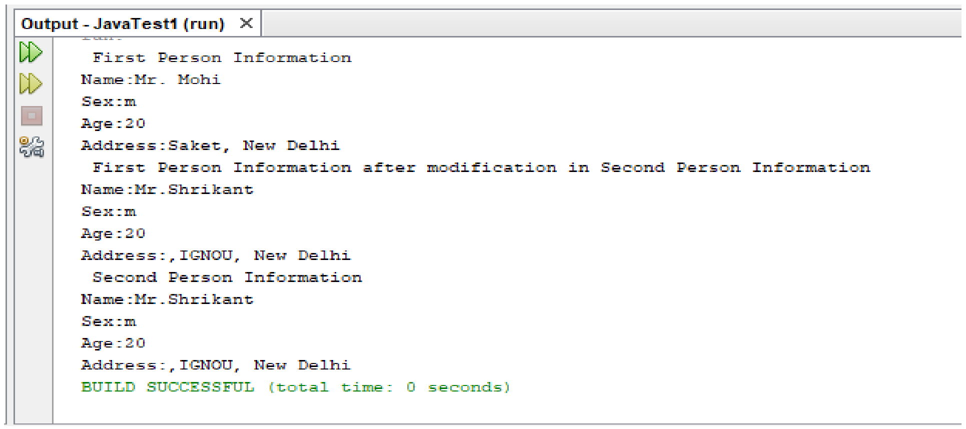
Now let us see how one object can be assigned to another object of the same type. Also, it is to note that an object assignment is different from a normal assignment. Let us see it with the help of a program in which a class Person with some data members and one member function Display.

Java Program

```
package javatest1;
class Person
{
String name;
int age ;
char sex;
String address;
void Display( )
{
System.out.println ("Name:"+name);
System.out.println ("Sex:"+sex);
System.out.println( "Age:"+age );
System.out.println ("Address:"+address);
}
}
public class JavaTest1
{
public static void main(String[] args)
{
Person First_P = new Person();
Person Second_P = new Person();
First_P.name= "Mr. Mohi";
First_P.sex = 'm';
First_P.age= 20;
First_P.address = "Saket, New Delhi";
System.out.println(" First Person Information");
First_P.Display();
Second_P = First_P;// Second_P refer to First_P
Second_P.name = "Mr.Shrikant";
Second_P.address = ",IGNOU, New Delhi";
System.out.println(" First Person Information after modification in Second Person
Information");
First_P.Display();//change in Second_P will have chane in First_P
System.out.println(" Second Person Information");
Second_P.Display();
}
```

}

Output:



Output - JavaTest1 (run)

```
 First Person Information
Name:Mr. Mohi
Sex:m
Age:20
Address:Saket, New Delhi
 First Person Information after modification in Second Person Information
Name:Mr.Shrikant
Sex:m
Age:20
Address:,IGNOU, New Delhi
 Second Person Information
Name:Mr.Shrikant
Sex:m
Age:20
Address:,IGNOU, New Delhi
BUILD SUCCESSFUL (total time: 0 seconds)
```

In this program, two objects, First_P and Second_P, are created. Object First_P is initialized with some values, and then the Display method is called on object First_P. It displays the value of data members of object First_P. Subsequently, object First_P is assigned to object Second_P as a reference variable. Here both objects First_P and object Second_P refer to the same object. Thus any change made in object First_P or Second_P will change the value in both the objects First_P and Second_P. You can see in the program that changes made in name and address by Second_P have also changed the values in First_P. Whenever this type of referencing of variables is used in a program, care should be taken while changing the values of instance variables of the object being referred.

Check Your Progress 1

1) Explain how objects are defined in Java.

……………………………………………………………………………………

……………………………………………………………………………………

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

……………………………………………………………………………………

……………………………………………………………………………………

3) What care should be taken in the case of object referencing?

……………………………………………………………………………………

……………………………………………………………………………………

……………………………………………………………………………………

4.3 INTRODUCING METHODS

From theprogrammer's point of view, a Java class is a group of values with a set of methods. The user of a class perform the functionality of the objects of that class only through the methods of that class. Methods are designed with the objective to fulfil a specific objective of large and complex calculations of the system. A method is created as a self-contained block of code which provide some specified function. Every method has a name, and the same method can be executed from many different objects, as per requirement. In Java, "a method describes the behavior of an object and is also a collection of statements that are grouped together to perform operations". Java program executes a method by calling it on objects. A method may or may not return a value. Also, the methods that do not return a value to be called without assigning them to any variables. When a method which returns a value is called, its returned value should be assigned to a variable. Alternatively, you may directly print the returned value. Also, you may use the returned value in the evaluation of an expression.

4.3.1 General Methods

In Java, a method is defined as follows:

```
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.

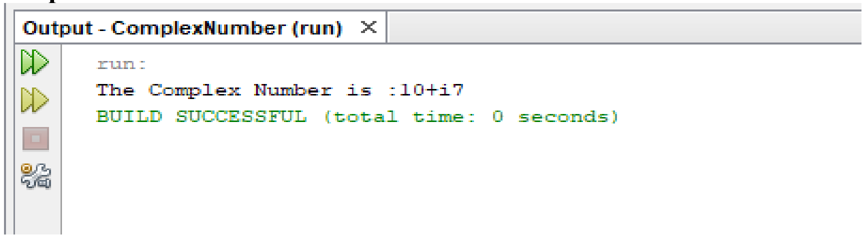
Now let us define a class to represent complex numbers. The complex class definition shown in the program given below illustrates how complex numbers can be represented. Two data member/variables *real* and *imag*, are declared. These represent the real and imaginary parts, respectively, of a complex number. In this program also, three methods, assignReal - assign values to the *real* variable and assignImag()- assign value to the *imaginary* variable and method showComplexl() shows the complex number respectively.

Java Program

```
package complexnumber;
class Complex
{
int real;
int imag;
void assignReal( int r)
{
real = r;
}
void assignImag( int i)
{
imag= i;
}
void showComplex ( )
{
System.out.println("The Complex Number is :"+ real +"+i"+imag);
}
}
public class ComplexNumber
{
```

```
public static void main(String[] args)
{
Complex R1 = new Complex();
R1.assignReal(10);
R1.assignImag(7);
R1.showComplex();
}
}
```

Output



Output - ComplexNumber (run)

```
run:
The Complex Number is :10+i7
BUILD SUCCESSFUL (total time: 0 seconds)
```

4.3.2 Static Methods

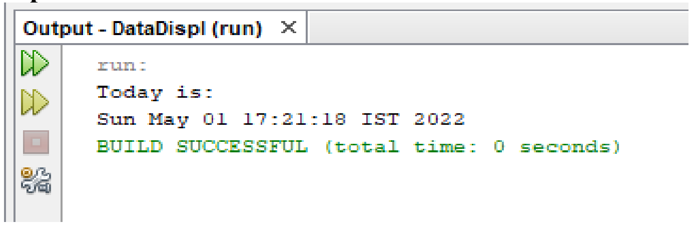
A static method is a characteristic of a class, not of the objects it has created.
To refer to instance methods and variables, you must instantiate the class first, then obtain the methods and variables from the instance. Static variables and methods are class variables or class methods since each class variable and each class method occur once per class. Non-static methods and variables belong to objects of the class. Instance methods and variables occur once per object(instance) of a class, or you can say every object has its own copy of instance variables. In a program, you may execute a static method without creating an object of the class. On the other hand, all other methods must be invoked using an object. Hence an object must exist before it can be used. As you have noticed, every Java application program has one main()method. This method is always static because Java starts execution from this method, and at that time, no object is created.

Let us see one example Java in which program:

Java Program

```
package datadispl;
import java.util.Date;
public class DataDispl
{
   public static void main(String[] args)
   {
      Date today = new Date();
      System.out.println("Today is:");
      System.out.println(today);
   }
  }
```

Output:



Output - DataDispl (run)

```
run:
Today is:
Sun May 01 17:21:18 IST 2022
BUILD SUCCESSFUL (total time: 0 seconds)
```

In this program, the last line of the main() method uses the **System** class from java.lang package to display the current date and time. See the line of code that invokes the *out.println ()* method.

System.out.println(today);

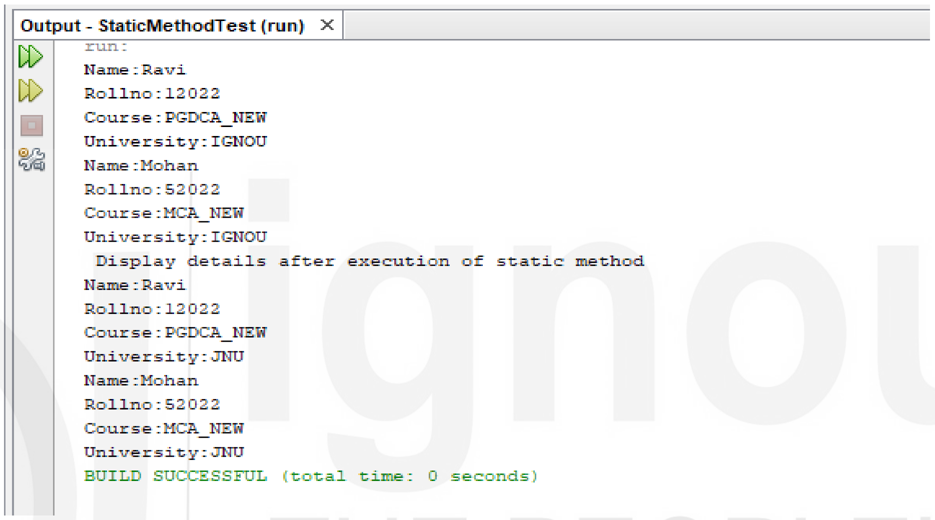
Now, look at the calling of *println()* method closely. When you call a static method, you write the class name instead of an object name.

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 name of the class with the name of the static method or static variable using dot (.) for accessing it. The point which should be noticed here is that the application in using method "**System.out.println",** 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. When you call a static method, you write the class name instead of an object name.

One more example of the use of static method is shown in the following program in which the variable name University is changed using static method as it is a common variable that belongs to Student class rather than individual objects of the Student class. Therefore change in one place will reflect in all the objects.

```
package staticmethodtest;
//Java Program, which demonstrates the use of a static method
class Student
{
 String Name;
 int Rollno;
 String Course;
 static String University = "IGNOU";
    //static method to change the value of static variable
 static void change_University()
  {
    University = "JNU";
  }
   //method to display values of students
 void display()
  {
   System.out.println("Name:"+Name);
   System.out.println("Rollno:"+Rollno);
   System.out.println("Course:" +Course);
   System.out.println("University:" +University);
  }
}
public class StaticMethodTest
{
public static void main(String args[])
{
Student S1 = new Student();
Student S2 = new Student();
S1.Name= "Ravi";
S1.Rollno =12022;
S1.Course = "PGDCA_NEW";
S2.Name= "Mohan";
S2.Rollno = 52022;
S2.Course = "MCA_NEW";
//calling display method  of Student Class
S1.display();
S2.display();
System.out.println(" Display details after execution of static method");
Student.change_University();//calling the static  change_University method  on class
name
S1.display();
S2.display();
 }
}
```

Output:



Output - StaticMethodTest (run)

```
run:
Name:Ravi
Rollno:12022
Course:PGDCA_NEW
University:IGNOU
Name:Mohan
Rollno:52022
Course:MCA_NEW
University:IGNOU
 Display details after execution of static method
Name:Ravi
Rollno:12022
Course:PGDCA_NEW
University:JNU
Name:Mohan
Rollno:52022
Course:MCA_NEW
University:JNU
BUILD SUCCESSFUL (total time: 0 seconds)
```

4.4 ENCAPSULATION

An *object* is *an* instance of a class. The class determines the features its objects will have and how those features are implemented in methods. A class encapsulates methods, data, and implementation of methods. You have already learned the basic concept of encapsulation in Unit 1 of this Block. The encapsulation is like a contract between the implementer of the class and the user of that class. Encapsulation help in hiding the details of how a piece of software works. A basic definition of encapsulation is "**the process of hiding all the details of a class definition that are not necessary to understanding how objects of the class are used**". In the context of programming, As a user of the software is not interested in the internal details of the system, a piece of software should be encapsulated so that only the necessary controls are visible and its details are hidden. Encapsulation is important as it simplifies the job of the programmer. It is important that for useful encapsulation, a class definition must be given in such a way that a programmer can use the class without seeing the hidden details. Encapsulation, when done correctly, neatly divides a class definition into two parts, which we call the *interface* and the *implementation*. The **class interface** tells programmers all they need to know to use the class in their programs. The class interface consists of the headings for the *public methods* and *public named constants* of the class, along with comments that tell to a user/programmer how to use these public methods and constants.

The **implementation** of a class consists of all of the private elements of the class definition, which includes the private instance variable of the class, along with the definition of both the public methods and private methods. Note that the class interface and the implementation of a class definition are not separate in your Java code but are mixed together.

Following are some of the basic guidelines to be used for defining a well-encapsulated class:

- Place a comment before the class definition that describes how the programmer should think about the class data and methods. For example, if the class describes an amount of money, the programmer should think in terms of rupee, dollars etc., and not in terms of how the class represents money.
- Declare all the instance variables in the class as private.
- Provide public accessor methods to retrieve the data by an object. Also, provide public methods for any other basic needs that a programmer will have for manipulating the data in the class.
- Try to give a comment before each public method to specify how to use the method.
- Make any helping methods use in the program as private.
- Write comments within the class definition to describe the implementation

Sometimes as a programmer, you may come up with a more efficient way to implement a method so that it may run faster. In such cases, you will need to change the implementation details of the class definition. When you have used the concept of encapsulation while defining your class, you should be able to go back and change the implementation details of the class definition without requiring changes in any program that uses the class (its methods). Also, as a learner this is a good way to test whether you have written a well-encapsulation class definition.

For example, in the case of online ticket booking systems for Train or Airlines, you may have a class for seat booking and payments. When there is any change in the rules for processing the ticket charge or payments, you need to modify the implementation without affecting the user experience.

4.5 ACCESS MODIFIERS

Access modifiers are keywords used to define the scope and behavior of the classes, methods and variables. You have seen the use of *public* keywords in many example Java programs so far. In Java, class definitions have different access levels. Access level determines which variables and methods in a class are available for access by the objects. In many cases, you need variables and code written for the internal consumption of the class and do not want to provide their access to the outside world. There are four possible access specifiers: *public, protected, private and default* access. About access specifiers, you will learn more in the next Block of this course. Below is the basic introduction to different access specifiers.

public: The *public* access refers to the data members and methods that can be freely accessed by code from any other class, except the class that inherits from this class and classes in the same package –which refers to a collection of classes – basically all the classes stored in the same directory.
protected: This access specifier provides controlled access to methods and variables outside the class. The *protected* methods and variables can be accessed by the same package or by any subclass (inherited class) outside of the package.
private: The *private* access means that the member will not be accessed by code in any other class, including classes that inherit from this one.
default: There is the fourth possibility, default access, which is used if no access specifier is mentioned.

Table 4.1 shows the access to members permitted by each modifier.

Modifier	**Generally Used by**	**Basic Description**
public	constructors, inner classes, methods and variables	This is most generic/liberal access level. Accessed by classes outside the package
protected	constructor, inner classes, methods and variables	This provide controlled access. Accessed by the same package or by any subclass outside of package
private	constructor, inner classes, methods and variables	This is most restrictive access level. Access only within class where they are declared.
No modifier (Default)	outer classes, inner classes, interface, constructor, method and variable	Access only within the package where they exist

Important point to remember that you cannot use more than one access modifier for a class, method or variable. Also, note that all the modifiers are optional, and it is up to the requirement of the programmer to use them as per the requirement of their design/solution. At the same time, it is a good programming style to use access specifiers as per need.

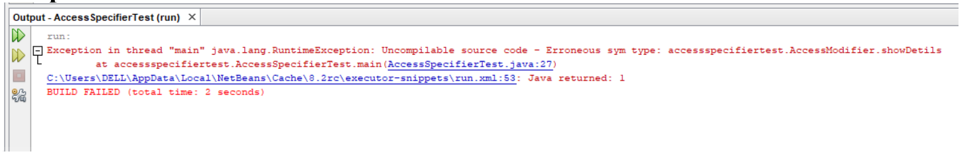
Need for Private Methods

When you are implementing a class, many a time you need to define all data fields *private* because public data are dangerous. Also, as a good practice, data access outside the class should be controlled to meet the basic philosophy of Object Orientation, which advocates for data protection. But why do we need private methods? In practice, most methods are public, but you need private methods for internal consumption of the class in certain circumstances. The private methods provide you the option to break up the code for computation into separate helper methods. Typically, these helper methods affect the functionality of public methods that work as the program's public interfaces. The private methods are implemented using *private* access specifier. You can see the below java program in which a private method named showDetails( ) is called by an object, and when you compile and run this program, as output, you will get compile-time error. This is because a private method can not be accessed from outside of the class. Similar test you may do by trying to access the private data members from outside of the class and observe the compiler error.

Java Program

```
package accessspecifiertest;
class AccessModifier
{
private String Name= "Shrikant";
private char Sex = 'M';
private int Age=16;
public void displayDetails ( )
{
System.out.println ("Name :" +Name);
System.out.println ("Sex :" +Sex);
System.out.println ("Age :" +Age);
}
private void showDetails( )
{
System.out.println ("Name :" +Name);
System.out.println ("Sex :" +Sex);
System.out.println ("Age :" +Age);
}
}
public class AccessSpecifierTest
{
public static void main (String args [] )
{
AccessModifier  AM=new AccessModifier ( );
//AM.displayDetails();
AM.showDetils ( );
 }
}
```

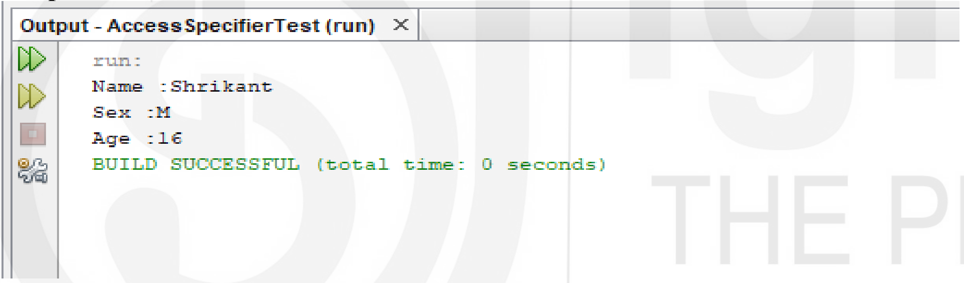
Output:



```
Output - AccessSpecifierTest (run)
run:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - Erroneous sym type: accessspecifiertest.AccessModifier.showDetils
        at accessspecifiertest.AccessSpecifierTest.main(AccessSpecifierTest.java:27)
C:\Users\DELL\AppData\Local\NetBeans\Cache\8.2rc\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 2 seconds)
```

Instead of calling
AM.showDetails();
in the above program if you call AM.displayDetails();, you will get following result.

Output:



```
Output - AccessSpecifierTest (run)
run:
Name  :Shrikant
Sex  :M
Age  :16
BUILD SUCCESSFUL (total time: 0 seconds)
```

4.6 CONSTRUCTORS IN JAVA

So far you have initialized the instance variables of the objects by assigning values to individual variables. Now we will discuss use of constructors for the initialization of objects(instance variables). A constructor is used for initialising the objects at the time of their creation. Constructors have the same name as the name of their class. Once a constructor is defined and used for object creation, it is automatically called, and the memory is allocated before the **new** operation completes. The constructor does not have any return type. It has an implicit return type of the class in which it is defined. In other words a constructor returns a class type. The job of a constructor is to initialize the instance variables of an object so that the created object is usable just after creation. For programmers, object construction is important, as it provides ready objects which can be used in solving the problem. Now let us see an example program, which we have used earlier in this unit and try to define a constructor for it.

In the following program class Complex does not have any constructor, and for initializing imaginary and real variables, two methods have been defined and used. Both the methods have been invoked on the object one by one for initializing the object's instance variables. In place of methods now, we will define a constructor and use it for initialization. We will make necessary changes in the class Complex. We will remove methods used for initializing variables and use one method having the same name of the class, i.e., Complex with two arguments.

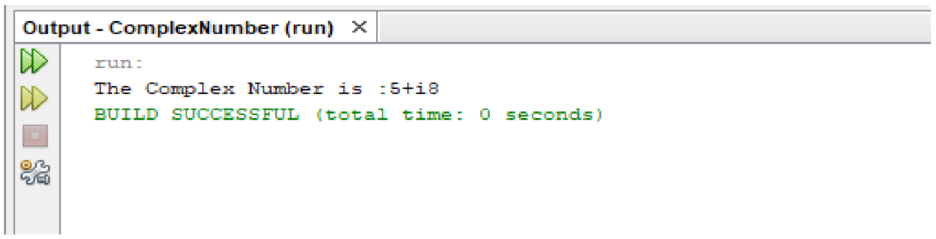
Original Java Program

```
package complexnumber;
class Complex
{
int real;
int imag;
void assignReal( int r)
{
real = r;
}
void assignImag( int i)
{
imag= i;
}
void showComplex ( )
{
System.out.println("The Complex Number is :"+ real +"+i"+imag);
}
}
public class ComplexNumber
{
public static void main(String[] args)
{
Complex R1 = new Complex();
R1.assignReal(10);
R1.assignImag(7);
R1.showComplex();
}
}
```

Modified Java Program in which constructor has beed defined

```
package complexnumber;
class Complex
{
int real;
int imag;
Complex( int r, int i) // defining constructor
{
real = r;
imag= i;
}
void showComplex ( )
{
System.out.println("The Complex Number is :"+ real +"+i"+imag);
}
}
public class ComplexNumber
{
public static void main(String[] args)
{
Complex R1 = new Complex(5,8);
R1.showComplex();
}
}
```

Output:



```
run:
The Complex Number is :5+i8
BUILD SUCCESSFUL (total time: 0 seconds)
```

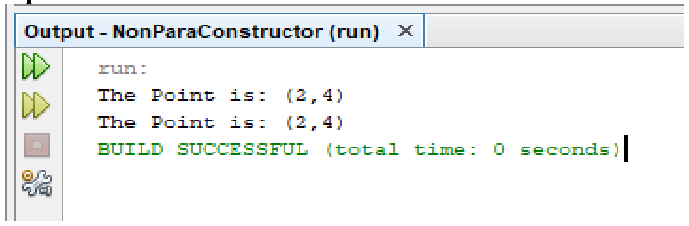
If you compare this program and the previous program, you will find that in this program, the instance variable of object R1 is initialized with the help of constructor Complex (5, 8), value 5 has been assigned to the real variable, and 8 has been assigned to the imag variable. In the previous program, to initialize these variables, methods assignReal() and assignImag() were used.

Further, if we look into the requirements of differet types of constructors, we can be defined in two ways. First a constructor which *may not have a parameter*. This type of constructor is known as *non-parameterized constructor*. The second, a constructor which 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 the following program in which objects of class Point is created using the non-parameterized constructor.

Example Java Program

```
package nonparaconstructor;
class Point
{
int i;
int j;
Point() // Non-parameterized constructor
{
i= 2;
j= 4;
}
void Display_Point()
{
System.out.println("The Point is: ("+i+","+j+")");
}
}
public class NonParaConstructor
{
public static void main( String args[])
{
Point  A1 = new Point();
Point A2 = new Point();
A1.Display_Point();
A2.Display_Point();
}
}
```

Output:



```
run:
The Point is: (2,4)
The Point is: (2,4)
BUILD SUCCESSFUL (total time: 0 seconds)
```

constructor Point( ) in above program is a non-parameterized constructor. Both the objects A1 and A2 are created by using the constructor Point( ). You can use this type of constructors to initialise those objects for which the initial value of instance variables is already known to you. When you need to give values of the instance variables at the time of object creation, use parameterized constructors.

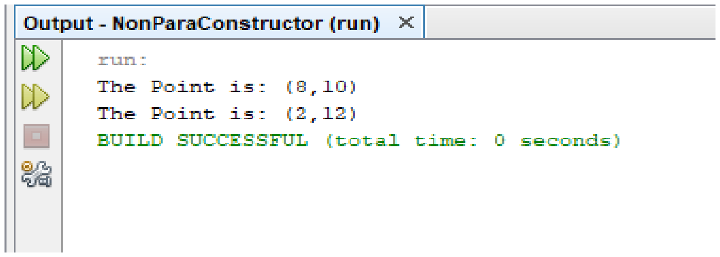
If you want to create objects of the class Point by giving your initial values of x and y coordinates, you can use parameterized constructor. You can do this by modifying the above program with the following changes.

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

Java Example Program

```
package nonparaconstructor;
class Point
{
int i;
int j;
Point( int a, int b) // Parameterized constructor
{
i= a;
j= b;
}
void Display_Point()
{
System.out.println("The Point is: ("+i+","+j+")");
}
}
public class NonParaConstructor
{
public static void main( String args[])
{
Point  A1 = new Point(8,10);
Point A2 = new Point(2,12);
A1.Display_Point();
A2.Display_Point();
}
}
```

Output:


Output - NonParaConstructor (run)

```
run:
The Point is: (8,10)
The Point is: (2,12)
BUILD SUCCESSFUL (total time: 0 seconds)
```

In this program, you can see that points A1 and A2 are initialized by values of programmer's choice by using parameterized constructor.

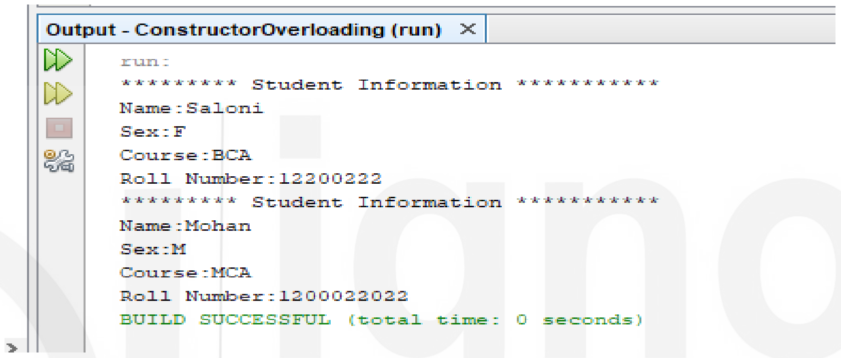
4.6.1 Overloading Constructors

You may ask a very valid question here: Can we have more than one constructor in a class? The answer is yes, you may have more than one constructor in a class. You have to keep all the constructors in a class by either having different types of arguments in the constructor or a different number of arguments in the constructors. This is essential because it would not be possible to identify which of the constructors is being invoked without this. These differences in the constructor's signature will help the compiler differentiate the constructors by considering the number of parameters passed to the constructor or the type of parameters passed to the constructors. "Having more than one constructor in a single class is known as constructor overloading". Constructor overloading enables the programmer to create objects with the desired number of attributes to be initialized. Basically, "constructor overloading is a technique in which a class can have any number of constructors that differ in parameter lists". In the program given below, in the class Student, more than one constructor are defined.

Example Java Program

```
package constructoroverloading;
class Student
{
String name;
int roll_no;
char sex;
String course;
Student( String n, int r, String c)
{
name = n;
roll_no  = r;
course = c;
sex = 'F';
}
Student(String n, int r, char c )
{
name  = n;
roll_no = r;
course = "MCA";
sex =c;
}
void Student_Info( )
{
System.out.println("********* Student Information **********");
System.out.println("Name:"+name);
System.out.println("Sex:"+sex);
System.out.println("Course:"+course);
System.out.println("Roll Number:"+roll_no);
}
}
public class ConstructorOverloading
{
public static void main(String[] args)
{
Student Student1 = new Student("Saloni", 12200222,"BCA");
Student Student2 = new Student("Mohan", 1200022022,'M');
Student1.Student_Info();
Student2.Student_Info();
}
}
```

Output:


Output - ConstructorOverloading (run)

```
run:
********* Student Information ***********
Name:Saloni
Sex:F
Course:BCA
Roll Number:12200222
********* Student Information ***********
Name:Mohan
Sex:M
Course:MCA
Roll Number:1200022022
BUILD SUCCESSFUL (total time: 0 seconds)
```

Both the constructors of this program can be used for creating two different objects. Here, different objects have different values assigned to instance variables of objects during their initialization through constructors. By using the constructor with three arguments three instance variables name, roll_no and course, are initialized. In another constructor, with the three arguments instance variables name and roll_no and sex are initialized. While calling, these constructors are identified by the type of arguments passed while constructor call.

You can observe that methods and constructors look similar, but there are some differences between them. These differences are given in the following table.

Table 4.2: Difference between constructor and method

Constructor	**Method**
A constructor is used to initialize an object.	A method is used to implement the behavior of an object.
The constructor must not have a return type.	The method must have a return type.
Constructor is invoked implicitly or explicitly.	A method is invoked explicitly.
The Java compiler provides a default constructor if there is no constructor.	The compiler does not provide a method in any case.
The constructor name must be the same as the class name.	In Java method name may or may not be the same as the class name. But it is not recommended to have the method name the same as the object.

Check Your Progress- 2

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

………………………………………………………………………………………

………………………………………………………………………………………

2) Write a program in Java to create SavingBankAccount class, which defines two different constructors to create objects and also defines a method to display details of the objects.

………………………………………………………………………………………

………………………………………………………………………………………

3) Run the following program and explain its output.

//Java Program

```
package boxttest;
class Box
{
int height;
int depth;
int length;
Box ( ) // Non parameterized constuctor
{
height=30;
depth=20;
length=10;
}
Box (int i, int j) //Parameterized constructor
{
height=i;
depth=j;
}
Box (int i, int j, int k) //Parameterized constructor
{
height=i;
depth=j;
length=k;
}
}
public class BoxtTest
{
public static void main (String args [ ] )
{
Box bx = new Box ( ) ;
System.out.println ("Depth of Box- bx : "+bx.depth);
System.out.println ("Height of Box- bx: "+bx.height);
System.out.println ("Lengtht of Box- bx: "+bx.length);
Box bx1 = new Box (10,15 ) ; // custructor call
System.out.println ("Height of Box- bx1:"+bx1.height);
System.out.println ("Depth of Box- bx1:"+bx1.depth);
System.out.println ("Length of Box- bx1: "+bx1.length);
Box bx2= new Box (12, 15, 20);//cunstrutor call
System.out.println ("Height of Box-bx2:"+bx2.height);
System.out.println ("Depth of Box- bx2:"+bx2.depth);
System.out.println ("Length of Box- bx2:"+bx2.length);
}
}
```

……………………………………………………………………………………

……………………………………………………………………………………

4.7 this KEYWORD

Java provides the keyword ***this*** that gives reference to the current object within the body of a method. By using ***this*** on the current object, programmers would be able to access the instance variables of that object. If a method wants to refer to the object through which it is invoked, it can refer to it by using ***this*** keyword. As a programmer, 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, have the same name(overlap) as the name of instance variables of the class, then 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, because ***this*** keyword can be used to refer to the objects directly.

Also, *this* keyword would allow the programmer to pass the current object as an argument to another method. The *this* keyword can be used anywhere the current object might appear. The dot(.) notation is used to refer to the object's instance variables.

Use of *this* keyword

- this.memberdata; // here memberdata is instance variable for *this*
- this. DisplayDetails(this); // method DisplayDetails(.) defined in this class will be called, and passes it to the current object.
- return this; // returns the current object

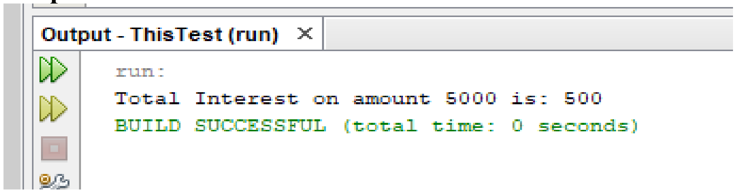
You can see this in the program given below. This program has one variable named ***rate*** and a method ***Total_Interest*** in the Account. The method ***Total_Interest*** is also having one local variable named ***rate***. To avoid conflict between both the **rate variables, *this*** keyword has been used with ***rate*** variable of the class Account.

Example Program

```
package thistest;
class Account
{
int rate ;
int amount;
int interest;
Account( int r, int a)
{
rate = r;
amount =a;
}
void Total_Interest( )
{
int rate = 5;
rate = this.rate+rate; // use of this keyword
interest = rate*amount/100;
System.out.println("Total Interest on amount "+amount+" is: "+interest);
}
}
public class ThisTest
```

```
{
 public static void main(String[] args)
{
Account  Acc1 = new Account( 5, 5000);
Acc1.Total_Interest();
}
}
```

Output:



```
Output - ThisTest (run)
run:
Total Interest on amount 5000 is: 500
BUILD SUCCESSFUL (total time: 0 seconds)
```

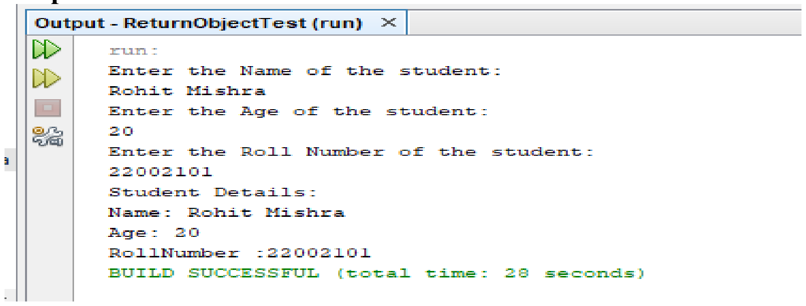
The following java program demonstrates how to return an object using this keyword. Last line of code in method SetValues() i.e *return.this* is returning the current object.

Example Java Program

```
package returnobjecttest;
import java.util.*;
class BCAStudent
{
   private String Name;
   private int Age;
   private int  RollNo;
   public BCAStudent SetValues() // method to set values of data members
   {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter the Name of the student: ");
      String name = sc.nextLine();
      System.out.println("Enter the Age of the student: ");
      int age = sc.nextInt();
      System.out.println("Enter the Roll Number of the student: ");
      int rollNo = sc.nextInt();
      this.Name = name;
      this.Age = age;
      this.RollNo = rollNo;
      return this;
   }
   public void DisplayDetails()
   {
      System.out.println("Name: "+Name);
      System.out.println("Age: "+Age);
      System.out.println("RollNumber :"+RollNo);
    }
}
public class ReturnObjectTest
{
  public static void main(String args[])
   {
      BCAStudent BCAObj = new BCAStudent();
      BCAObj = BCAObj.SetValues();
      System.out.println("Student Details:");
      BCAObj.DisplayDetails();
   }
```

}

Output



```
Output - ReturnObjectTest (run)
run:
Enter the Name of the student:
Rohit Mishra
Enter the Age of the student:
20
Enter the Roll Number of the student:
22002101
Student Details:
Name: Rohit Mishra
Age: 20
RollNumber :22002101
BUILD SUCCESSFUL (total time: 28 seconds)
```

4.8 USING OBJECTS AS PARAMETERS

Sometimes you need to pass objects as a parameter to methods, and also, objects may be returned from the methods. Let us look at these two aspects one by one.

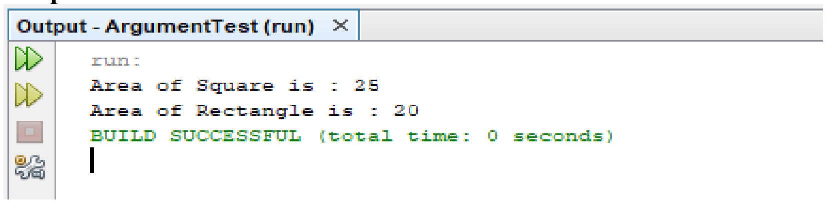
4.8.1 Object Passed as Argument in Methods

During problem-solving, when you define methods, sometimes you need to pass objects as arguments so that you may generalize a method. These generalized methods can be used for performing operations on a variety of data.In the program given below for finding the area of objects square and rectangle, the methods Area_Square and Area_Rectangle for finding the area of square and rectangle respectively are defined. In this program, primitive variables are passed as an argument rather than objects.

Java Program

```
package argumenttest;
class Area
{
int Area_Square( int i)
{
 return i*i;
}
int Area_Rectangle(int a,int b)
{
 return a*b;
}
}
public class ArgumentTest
{
public static void main(String args[])
{
 Area a = new Area();
 int area;
 area = a.Area_Square(5);
 System.out.println("Area of Square is : "+area);
 area = a.Area_Rectangle(5,4);
 System.out.println("Area of Rectangle is : "+area);
}
}
```

Output:



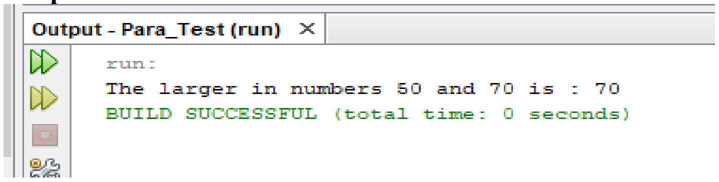
```
run:
Area of Square is : 25
Area of Rectangle is : 20
BUILD SUCCESSFUL (total time: 0 seconds)
```

You can see that passing parameters in methods Area_S and Area_R is *call by value*. In your first semester, you have already studied two basic ways call by value and call by reference of parameter passing in functions in the course MCS 201. 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. In Java everything except the absolute value is passed by reference. You can see in the program given below in the method max, that the two parameters are passed by value. The values of variables are of primitive type.

Example Program

```
package para_test;
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 = 50, num2 = 70, num3;
num3 = max( num1, num2);
System.out.println("The larger in numbers "+num1 +" and "+num2+" is : "+num3);
}
}
```

Output:



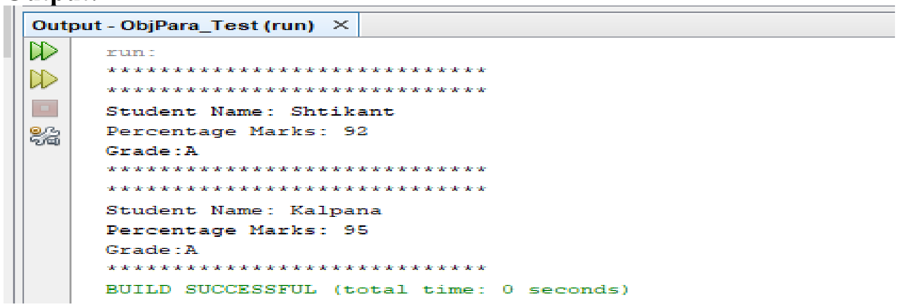
```
run:
The larger in numbers 50 and 70 is : 70
BUILD SUCCESSFUL (total time: 0 seconds)
```

The objects passing as a parameter in Java are called variable passing by reference. As mentioned earlier in this section in Java, we can pass a reference of the object to the formal parameter in the methods. In such a case, any changes to the local object inside the method will modify the object that was passed to the method as argument. In the program given below to method Set_Grade , object of the class Marks is passed as argument. In this method, the instance variable *grade* of the object passed is assigned some value, and this assignment of value changes the object itself.

Java Program:

```
package objpara_test;
class Marks
{
String name;
int percentage;
String grade;
Marks(String n, int m)
{
name = n;
percentage = m;
}
void Display()
{
System.out.println("******************************");
System.out.println("Student Name: "+name);
System.out.println("Percentage Marks: "+percentage);
System.out.println("Grade:"+grade);
System.out.println("******************************");
}
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";
}
}
public class ObjPara_Test
{
public static void main( String a[])
{
Marks ob1 = new Marks("Shtikant",92);
Marks ob2 = new Marks("Kalpana",95);
Marks.Set_Grade(ob1);
System.out.println("******************************");
ob1.Display();
Marks.Set_Grade(ob2);
ob2.Display();
}
}
```

Output:



Output - ObjPara_Test (run)

```
run:
******************************
******************************
Student Name: Shtikant
Percentage Marks: 92
Grade:A
******************************
******************************
Student Name: Kalpana
Percentage Marks: 95
Grade:A
******************************
BUILD SUCCESSFUL (total time: 0 seconds)
```

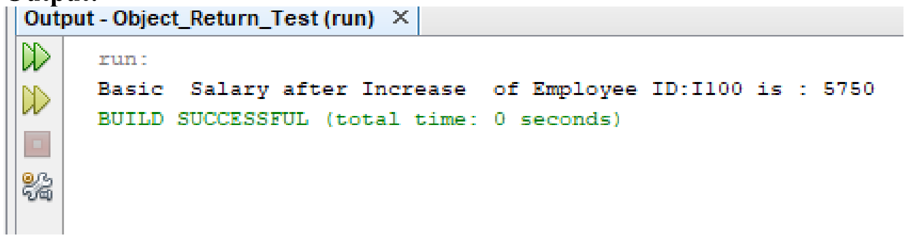
4.8.2 Returning Objects

Similar to other primitive data types, in Java a method can return data of class type, i.e. object of a class. When such a method is called, an object returned from that method can be stored in any other object of that class, It is similar to the value of basic type returned is stored in a variable of that data type. You can see in the program below where an object of class Salary is returned by method Increae_Salary. Also you may notice that to this method object is passed as an argument.

Salary Increase_Salary (Salary s) // object is passed as argument and also object is returned from the method.

```
package object_return_test;
class Salary
{
int basic ;
String Emp_id;
Salary( String a, int b)
{
Emp_id = a;
basic = b;
}
Salary  Increase_Salary ( Salary s )
{
s.basic = basic*115/100;
return s;
}
}
public class Object_Return_Test
{
public static void main(String[] args)
{
Salary s1 = new Salary("I100",5000);
Salary s2; // A new salary object
s2 = s1.Increase_Salary( s1);
System.out.println("Basic  Salary after Increase  of Employee ID:"+ s2.Emp_id +" is :
" + s2.basic);
}
}
```

Output:



```
run:
Basic  Salary after Increase  of Employee ID:I100 is : 5750
BUILD SUCCESSFUL (total time: 0 seconds)
```

4.9 GARBAGE COLLECTION

One very important feature of Java is garbage-collection. It is also known as a garbage-collected heap. In the garbage collection process, Java free the dynamically allocated memory which is no longer referenced. Java takes care of reclaiming memory that is no longer in use by garbage collection. It shields the substantial complexity of memory allocation and the garbage collection process from the programmer. Because the heap is garbage-collected, Java programmers do not have to explicitly free the allocated memory. In Java "*new*" operator allocate memory to object at the time of creation on the heap at run time. The JVM's heap stores all the objects created by an executing Java program. In any program when an object no longer in reference, the heap space it occupies must be freed so that the space is available for new objects in the program or system.

Advantages of garbage collection

Giving the job of garbage collection to the JVM has several advantages. Two main advantages are:
1. First, it can make programmers more productive.
2. Second advantage of garbage collection is that it ensure program integrity.

Disadvantages of garbage collection

One major disadvantage of garbage collection is that it adds overhead to the system, which adversely affects program performance. As 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 eat up more CPU time than would have been required when the program explicitly freed unrefined memory. The second disadvantage of garbage collection is that "the programmers in a garbage-collected environment have less control over the scheduling of CPU time" as it is devoted to freeing objects that are no longer needed.

The JVM has a low-priority thread that checks the graph of objects to find orphaned objects. In the process, the objects those are found are marked as available to be collected. A separate process is run by JVM, when necessary, to reclaim that memory.

1.4.9.1 the finalize ( ) method

You can request garbage collection by calling System.gc( ). Calling this method is merely a request for garbage collection. It may or may have any effect; it is not guaranteed to run when one asks for garbage collection.

Sometimes some objects have to take independent resources therefore before the garbage collector takes the object. The programmer needs to have provision take independent resources from those objects before garbage collection of that object. To perform this operation, finalized () method is used. The finalized () is called by the garbage collector when it determines no more references to the object exist.

fnalized() 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.

You can specify code to run when an object is collected by writing a finalize( ) method in a class. There is no guarantee that this method will ever run. The advantages of garbage collection are.

- The programmer does not need to worry about dereferencing an object.
- It is done automatically by the JVM.
- Increases memory efficiency and decreases the chances of memory leak.

One can add a finalize method to any class. The finalize method will be called before the garbage collector sweeps away the object.

The syntax for finalize () method is as follows:

```
protected void finalize ( )
 {
// finalize-code
}
```

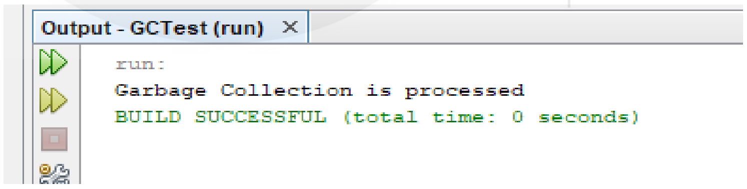
gc ( ) method

The static gc ( ) method is defined in the System class to encourage th JVM to do some garbage collecting and recover the memory that the objects occupy using System.gc ( ). This is a best-effort deal on the part of the JVM for memory management. When the gc ( ) method returns, the JVM has tried to reclaim the space occupied by discarded objects, but there is no guarantee that it has all been recovered. It totally depends on the processing of JVM.

Java Program

```
package gctest;
public class GCTest
{
public static void main (String [ ] args )
{
        GCTest  Obj = new GCTest ( );
        Obj= null;
        System. gc ( );
}
protected void finalize ()
{
System.out.println ("Garbage Collection is processed");
}
}
```

Output:



Output - GCTest (run)

```
run:
Garbage Collection is processed
BUILD SUCCESSFUL (total time: 0 seconds)
```

Check Your Progress- 3

1) Run the following code and show the result.

```
package usethis;
class Student
 {
String Name;
int RollNo;
String Course;
Student (String Name,int RollNo, String Course)
 {
```

ignou THE PEOPLE'S UNIVERSITY

```
this.Name = Name;
this.RollNo = RollNo;
this.Course = Course;
}
void DisplayDetails ( )
{
System.out.println (" Name of Student:" + Name);
System.out.println ("Student Rellnumber:" + RollNo);
System.out.println ("Course Enrolled:" + Course);
 }
}
public class UseThis
{
public static void main (String args [ ])
{
Student s1 = new Student ( "Manish",202022,"MCA") ;
Student s2 = new Student ("Akshay",202221,"BCA") ;
System.out.println("************Student 1 Details***************");
s1.DisplayDetails ( ) ;
System.out.println("************Student 2 Details***************");
s2.DisplayDetails ( ) ;
}
}
```

…………………………………………………………………………..

…………………………………………………………………………

2) Run the following code and show the result.

```
package returnobjtest;
class Shape
{
int length;
int breadth;
Shape(int i, int j)
{
length = i;
breadth = j;
}
Shape getShapeObject(Shape s )
{
Shape spo;
spo=s;
return spo;
}
}
public class ReturnObjTest
{
public static void main (String args [ ] )
{
Shape ob1 = new Shape(40, 50);
Shape ob2;
ob2 = ob1.getShapeObject(ob1 ) ;
System.out.println ("ob1.length: "+ ob1.length);
System.out.println ("ob1.breadth: "+ ob1.breadth);
System.out.println ("ob2.lenght:" + ob2.length);
System.out.println ("ob2.breadth:"+ ob2.breadth);
}
```

}

3) What is advantage of garbage collection?

……………………………………………………………………………………

……………………………………………………………………………………

4.10 SUMMARY

This unit explained defining classes and methods in Java. It has been demonstrated how to assign objects as a reference variable. Every object has its state and behavior. Objects behavior are defined inside the class in terms of member functions. Different types of methods, such as general and static methods, have been explained. The concept of encapsulation has been explained in this unit with example. The use of different access specifiers in Java programming is demonstrated with examples. This unit also discussed the defining constructors with the help of example programs. This unit explained arguments passing to the function. Towards the end, this unit explains the garbage collection feature of Java.

1.11 SOLUTIONS/ANSWERS

Check Your Progress - 1

1) Object definition in Java is a two-step process:

   i Declare variable of Account type (class type).

   Account Acc // declaration of object named Acc of type Account

   ii. Allocate required space to the object using new operator:
   Acc = new Account(); // allocating memory.

2) Objects are the basic elements of a program used for solving the problem. Objects are variables of class type and known as instance of classe. Objects should be considered a black box with public interfaces to access their functionality. Interfaces provide the functionality with the help of member functions.

   In object oriented programming, data is given importance. Member functions and member data are kept inside the class to restrict access of data members and member functions from outside of the class or avoid accidental modification. Data hiding is ensured by keeping data inside the class. Definition of member functions/methods ares kept inside the class, which provides the scope of modification in the definition of a method without affecting the outside world.

3) In Java one object can be used as a reference to another object of the same type i.e. both are of the same class type. When objects are used as a reference, care should betake in performing operations on objects because if there is any change invalues data members of one object, values of respective data members of the second object also get changed.

Check Your Progress - 2

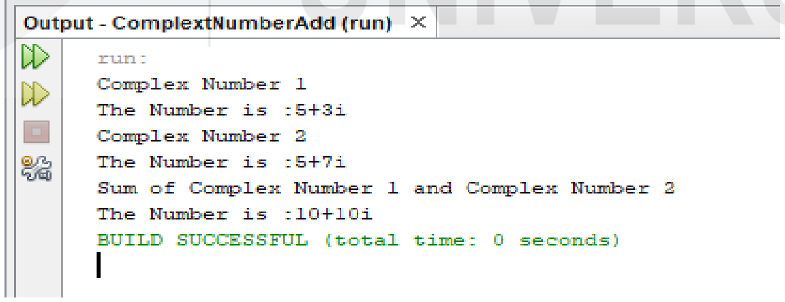
1) To intialise the objects, constructors are used in Java programs. Using constructors, you get the objects which are ready to use. Following is a Java program in which objects of Complex class are created using constructor defined in this class.

Java Program

```
package complextnumberadd;

//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 ComplextNumberAdd
{
public static void main(String args[])
{
Complex C1= new Complex(5,3);
Complex C2 = new Complex (5,7);
Complex C3 = new Complex (0,0);
System.out.println("Complex Number 1");
C1.ShowNumber();
System.out.println("Complex Number 2");
C2.ShowNumber();
C3.real = C1.real +C2.real;
C3.imaginary = C1.imaginary +C2.imaginary;
System.out.println("Sum of Complex Number 1 and Complex Number 2");
C3.ShowNumber();
}
}
```

Output:

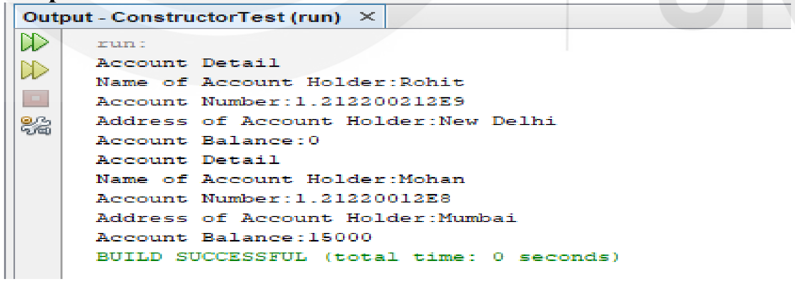


```
run:
Complex Number 1
The Number is :5+3i
Complex Number 2
The Number is :5+7i
Sum of Complex Number 1 and Complex Number 2
The Number is :10+10i
BUILD SUCCESSFUL (total time: 0 seconds)
```

2) In this program there are two different constructors of SavingBankAccount class are defined. The method Display_Details( ) is defined to display details of the objects.

```
package constructortest;
class SavingBankAccount
{
 private String Name;
 private double  Account_No;
 private String Address;
 private int  Balance;
SavingBankAccount(String n, double a, String addr)
{
Name =  n;
Account_No = a;
Address = addr;
}
SavingBankAccount (String n, String addr, double a , int b)
{
Name =  n;
Address = addr;
Account_No = a;
Balance = b;
}
void Display_Details()
{
System.out.println("Account Detail");
System.out.println("Name of Account Holder:"+Name);
System.out.println("Account Number:"+Account_No);
System.out.println("Address of Account Holder:"+Address);
System.out.println("Account Balance:"+Balance);
}
}
public class ConstructorTest
{
public static void main( String args[])
{
SavingBankAccount  AC1  =  new  SavingBankAccount("Rohit",  1212200212,"New
Delhi");
SavingBankAccount   AC2   =   new   SavingBankAccount(   "Mohan",   "Mumbai",
121220012,15000);
AC1.Display_Details();
AC2.Display_Details();
}
}
```

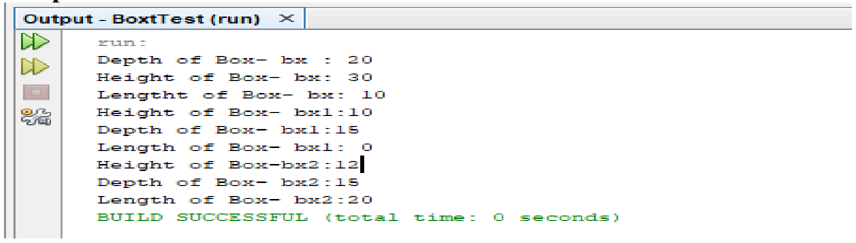
Output:


Output - ConstructorTest (run)

```
run:
Account Detail
Name of Account Holder:Rohit
Account Number:1.212200212E9
Address of Account Holder:New Delhi
Account Balance:0
Account Detail
Name of Account Holder:Mohan
Account Number:1.21220012E8
Address of Account Holder:Mumbai
Account Balance:15000
BUILD SUCCESSFUL (total time: 0 seconds)
```

3)

Output:


Output - BoxtTest (run)

```
run:
Depth of Box- bx : 20
Height of Box- bx: 30
Lengtht of Box- bx: 10
Height of Box- bx1:10
Depth of Box- bx1:15
Length of Box- bx1: 0
Height of Box-bx2:12
Depth of Box- bx2:15
Length of Box- bx2:20
BUILD SUCCESSFUL (total time: 0 seconds)
```

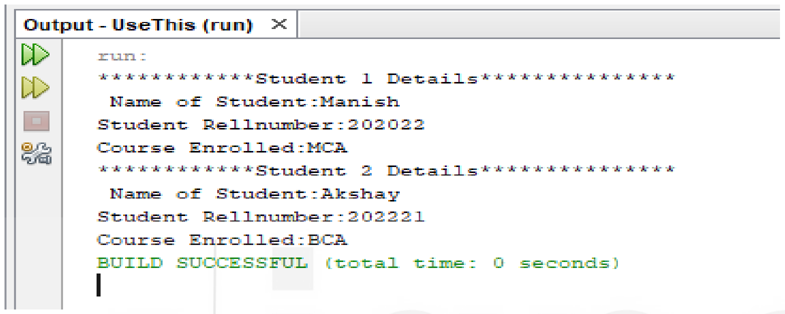
The calling of different constructors is demonstrated in the program. For creating object bx1 when the constructor with two variables is called, the variable *length* is assigned the default value.

Check Your Progress- 3

1)

Output:

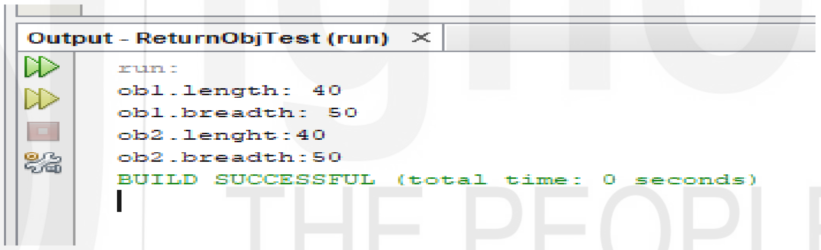
The following output demonstrates the creation of objects using constructors and their use in the program.



```
run:
************Student 1 Details***************
 Name of Student:Manish
Student Rellnumber:202022
Course Enrolled:MCA
************Student 2 Details***************
 Name of Student:Akshay
Student Rellnumber:202221
Course Enrolled:BCA
BUILD SUCCESSFUL (total time: 0 seconds)
```

2)

Ouptput:



```
run:
ob1.length: 40
ob1.breadth: 50
ob2.lenght:40
ob2.breadth:50
BUILD SUCCESSFUL (total time: 0 seconds)
```

Above is the output of the program, which demonstrates how an object is returned from a method.

3) There are two major advantages of automatic garbage collection

i. Increase programmer productivity.

ii. Ensure program integrity.

4.12 REFERENCES/FURTHER READINGS

- Herbert Schildt, "Java The Complete Reference", McGraw-Hill, 2017.
- Savitch, Walter, " Java: An introduction to problem solving & programming", Pearson Education Limited, 2019.
- S.Sagayaraj,R. Denis, P.Karthik and D.Gajalakshmi, "Java Programming for Core and Advanced Learners", University Press, 2018.
- Neil, O. , "Teach yourself JAVA", Tata McGraw-Hill Education, 1999.