Unit – 3 Objects and Classes

3.1 Introduction – Classes and Objects

3.2 Data members, methods

3.3 Types of Constructors

3.4 Overloading

3.5 Packages

3.6 Access modifier

3.7 Inner classes

Prof. A. P. Chaudhari (M.Sc., SET)
HOD, Department of Computer Science
SVS's Dadasaheb Rawal College, Dondaicha

3.1 Introduction – Classes and Objects

Classes:

A class is a user defined data type, it defines a new data type. Once defined, it can be used to create objects of that type. A class definition is a describes what pieces of information the new type holds and methods with which that information can be manipulated, so the class is composed of fields and methods. A field is a variable defined within a class definition that is associated with an instance (object) of that class. A method is a set of Java statements which can be included inside a Java class. A class with only data fields has no life. Objects created by such a class can not respond to any massages. A class is declared by the keyword ‘class’.

3.1 Introduction – Classes and Objects

Syntax:- class class_name

{

variable declaration;

method declaration;

}

e.g:

```
class employee
{
        int  empid;
        float salary;
}
```

Here class employee contains two data members empid and salary.

3.1 Introduction – Classes and Objects

Objects:

Object is an instance (variable) of class. The data in a class is in the form of instance variables. We can declare the instance variables exactly the same way as we declare the local variables. These variables are also called as data members. There are two steps to create an object from a class:

1) Declaration:

We have to specify what type (i.e. Class) the object will be. A variable declaration with a variable name with an object type.

Syntax: class_name object_name;

Where class_name is the name of already defined class and the object_name is a valid identifier.

3.1 Introduction – Classes and Objects

2) Instantiation (Creating Objects):

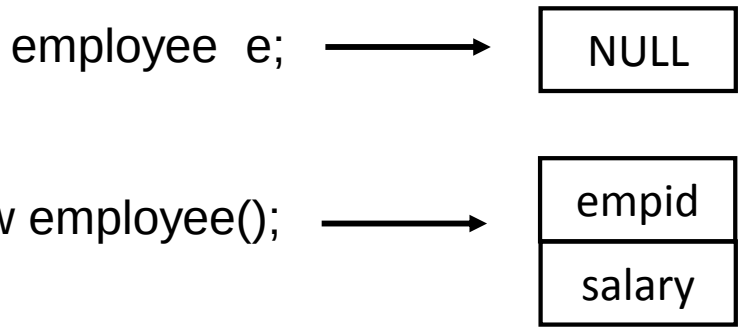
Objects are created using the 'new' keyword. The 'new' keywords creates an object of the specified class and returns the reference of that object.

Syntax: object_name = new class_name([arguments]);

e.g.: Let us create an object of above employee class:

```
employee  e;
e = new employee();
```

Following figure shows the above process:



3.2 Data members, Methods

The data members or variables defined within a class are called as instance variables because each instance of the class (i.e. each object) contains its own separate copy of variables. The object created by a class having only variables cannot communicate effectively with external world.

Therefore, methods are necessary for manipulating the data contained in the class. A Java methods is an collection of statements that are grouped together to perform an operation. The general form of declaration od methods is:

```
data_type  method_name(arguments_list)
{
        body_of _method
}
```

The method declaration contains four different parts:

1) Data_type of values returned by method (data_type)
2) Name of the method (method_name)
3) The list of arguments (argument_list)
4) Body of the method

3.2 Data members, Methods

e.g:

```
class employee
{
        int empid;
        float salary;
        void getdata(int a, float b)
        {
                empid = a;
                salary = b;
        }
}
```

Here empid and salary are the data members of a class employee and getdata() is the method of class.

3.2 Data members, Methods

```
import java.io.*;
class book
{
        String title;
        float price;
        void getbook(String name, float rate)
        {
                title = name;
                price = rate;
        }
        void putbook()
        {
                System.out.println("Title: "+title);
                System.out.println("Price: "+price);
        }
}
class bookdemo
{
        public static void main(String args[])throws IOException
        {
                book b1 = new book();
                DataInputStream d = new DataInputStream(System.in);
                System.out.println("Enter Title:");
                String t = d.readLine();

                System.out.println("Enter Price:");
                float p = Float.parseFloat(d.readLine());

                b1.getbook(t,p);
                b1.putbook();
        }
}
```

OutPut:

```
Enter Title: JAVA
Enter Price: 240
Title: JAVA
Price: 240
```

3.3 Types of Constructors:

Constructor is a special function which has the same name as class itself. It is automatically called when an object it created. Constructor must have no explicit return type. Constructors may be private, protected or public. Multiple constructors may be exist, but they must have different signatures. i.e. different numbers and types of input parameters.

Types of Constructors:

1) Default Constructor
2) Parameterized Constructor

1) Default Constructor:

A constructor that has no parameters is known as default constructor. In other words, when the object is created, java creates a no argument constructor automatically known as default constructor. Default constructor provides the default values to the object like zero (0), Null, etc. depending on that type.

3.3 Types of Constructors:

```
import java.io.*;
class area
{
        float pi, red, a;
        area()
        {
                pi = 3.14f;
                System.out.println("Default
                        Constructor executed");
        }
        void getred(float r)
        {
                red = r;
        }
        void cal()
        {
                a = pi*red*red;
                System.out.println("Area is:"+a);
        }
}
```

```
class circle
{
        public static void main(String args[])
        {
                area a1 = new area();
                a1.getred(4.5f);
                a1.cal();
        }
}
```

Output:

```
Default Constructor executed
Area is:63.585
```

3.3 Types of Constructors:

```
import java.io.*;
class student
{
        int rollno;
        String name;
        String std;
        student()
        {
                std = "T.Y.B.Sc.";
        }
        void getdata(int rno, String title)
        {
                rollno = rno;
                name = title;
        }
        void putdata()
        {
           System.out.println("Roll Number:"+rollno);
           System.out.println("Name:"+name);
           System.out.println("Standard:"+std);
        }
}
class sinfo
{
        public static void main(String args[])
        {
                student s1 = new student();
                s1.getdata(3707,"Rohit Patil");
                s1.putdata();
        }
}
```

Output:
Roll Number:3707
Name:Rohit Patil
Standard:T.Y.B.Sc.

3.3 Types of Constructors:

2) Parameterized Constructor:

A constructor that has parameters or arguments is known as parameterized constructor. It is used to provide different values to the distinct objects. With the use of this constructor, it is possible to initialize objects with different set of values at the time of their creation. These different set of values initialized to objects must be passed as arguments when constructor is invoked.

```
import java.io.*;
class area
{
    float pi, red, a;
    area(float r)
    {
        red = r;
        pi = 3.14f;
        System.out.println("Default
    Constructor executed");
    }
    void cal()
    {
        a = pi*red*red;
        System.out.println("Area is:"+a);
    }
}
class circle1
{
    public static void main(String args[])
    {
        area a1 = new area(4.5f);
        a1.cal();
} }
```

3.4 Overloading:

In Java overloading allows different methods to have same name, but different signatures where signatures can differ by number of input parameters or type of input parameters or both. Overloading is related to compile time polymorphism.

When we call a method the java compiler first compare the method name then number of arguments and their data types to decide which method is to be called.

3.4 Overloading:

```
import java.io.*;
class addition
{
        int add(int x, int y)
        {
                return(x+y);
        }
        float add(float x, float y)
        {
                return(x+y);
        }
        int add(int x, int y, int z)
        {
                return(x+y+z);
        }
        float add(int x, float y)
        {
                return(x+y);
        }
        float add(float x, int y)
        {
                return(x+y);
        }
}
```

```
class mover
{
        public static void main(String args[])
        {
                addition a = new addition();
                System.out.println(a.add(20,30,50));
                System.out.println(a.add(18.7f, 15));
                System.out.println(a.add(30,25));
                System.out.println(a.add(28.9f, 33.3f));
                System.out.println(a.add(81, 62.5f));
        }
}
```

Output:

```
100
33.7
55
62.199997
143.5
```

3.5 Packages:

Package is called as the collection of classes and interfaces. A package can be defined as a grouping of related or similar types of classes, interfaces and sub packages providing access protection and namespace management.

Advantages of packages:

- Package is used to categorized the classes and interfaces so that they can be easily maintained.
- Package provides access protection.
- Easy to locate the files.
- Reusability of code is one of the most important requirements in the software industry. Reusability saves time, effort and also ensures consistency.

3.5 Packages:

Types of Packages:

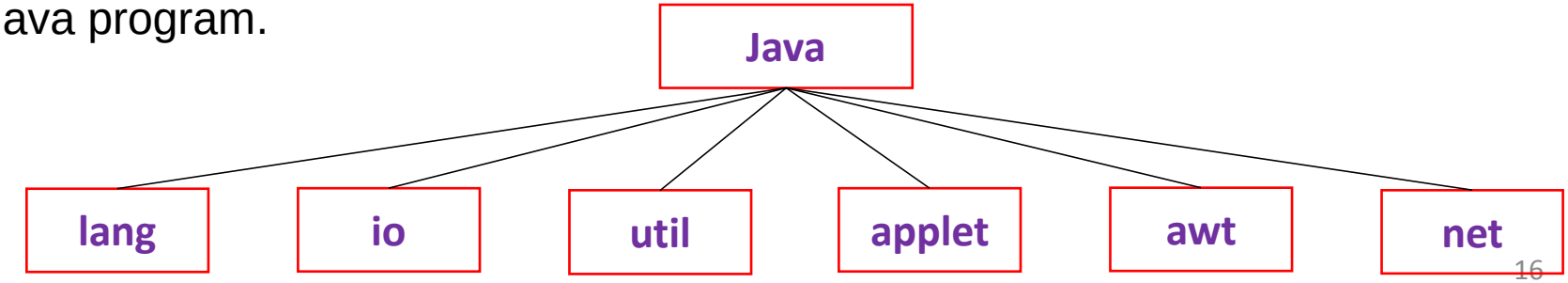
Packages are classified into two types-

1) Built in Packages (Java API Packages)
2) User Defined Packages

1) Built in Packages:

Java API (Application Program Interface) library provides a large number of classes grouped into different packages according to different functionality.

Following figure shows the built in packages which are frequently used in a Java program.



3.5 Packages:

1) java.lang:

It contains language support classes such as System, Tread, Exception, etc. This package automatically imported.

2) java.io:

It contains classes for supporting input / output operations.

3) java.util:

It contains utility classes such as Vector, Arrays, Linked List, Stack, etc. It also supports for date-time operations.

4) java.applet:

It contains classes for creating and implementing applets.

5) java.awt: (abstract window toolkit)

It contains classes using GUI such as Window, Frame, Panel, etc.

6) java.net: It contains classes for networking operations.

3.5 Packages:

All packages contain their own classes and classes contain their own methods. There are two ways of accessing the classes from packages. The first is to use the fully qualified name of the class that we want to use in the program. This is done by using a package name containing the class and then appending the class name to it using a dot operator (.).

e.g: java.util.stack;

To import the specified class in the specified package in the source file as follows:

```
import packagename.classname;
```

OR

```
import packagename.*;
```

e.g: `import java.io.*;`

3.5 Packages:

2) User defined packages:

This packages are defined by the user. While creating a package we must first declare the name of the package using the reserve word 'package'. This must be the first statement in a java source file. Then define a class just as we normally define it.

Syntax:

```
package packagename;
public class classname
{
        body_of_class;
}
```

e.g:

```
package mypackage1;
public class demo1
{
        body_of_class;
}
```

Here, package name is mypackage1. The class demo1 is considered as the part of this package. This code must be save as demo1.java and must be located in directory (folder) name mypackage1.

3.5 Packages:

Steps for creating a user defined package:-

Step 1: Create a new folder with package name as follows:

My Computer → C Drive → Jdk1.4 → Bin → New Folder

Rename New Folder as mypackage1

Step 2: Open a command prompt;

Click on start button → search cmd → enter

Step 3: Open the editor for creating a package:

C:\user\documents>cd..

C:\user>cd..

C:\>cd jdk1.4

C:\jdk1.4>cd bin

C:\jdk1.4\bin>cd mypackage1

C:\jdk1.4\bin\mypackage1>edit

3.5 Packages:

Step 4: Write a program for mypackage1 as follows:

```
package mypakage1;
public class demo1
{
        public int i = 10;
        public void disp1()
        {
                System.out.println("Value of i is:"+i);
        }
}
```

Step 5: Save the above program as demo1.java in mypackage1 folder.

Step 6: Exit from the editor and command editor appear.

Step 7: For creating a package **mypackage2** same as above steps 1,2,3.

3.5 Packages:

Step 8: Write a program for mypackage2 as follows:

```
package mypackage2;
public class demo2
{
        public int j =20;
        public void disp2()
        {
                System.out.println("Value of j is:"+j);
        }
}
```

Step 9: Save the above program as demo2.java in mypackage2 folder.

Step 10: Exit from editor and command prompt appear.

3.5 Packages:

Step 11: Now, we import the above both packages in a program as follows:

(New program create in a C:\jdk1.4\bin)

```
import mypackage1.*;
import mypackage2.demo2;
class packdemo
{
        public static void main(String args[])
        {
                demo1  d1 = new demo1();
                d1.disp1();
                demo2  d2 = new demo2();
                d2.disp2();
                System.out.println("Addition of i and j:"+(d1.i+d2.j));
        }
}
```

Output:

```
Value of i is: 10
Value of j is: 20
Addition of i and j: 30
```

3.6 Access Modifier:

Java provides a number of access modifiers to restricts the scope of class, constructor, variables, methods or data member. Access modifiers are also known as visibility modifiers.

There are four types of access modifiers available in Java:

1) **Default:-** No modifiers are needed
2) **Private:-** Visible to the class only
3) **Public:-** Visible to the world
4) **Protected:-** Visible to the package and all subclass

1) Default Access Modifier: If no access specifier is used then default specifier is used by Java compiler. Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc. A variable or method declared without any access modifier is available to any other class in the same package.

3.6 Access Modifier:

2) Private Access Modifier: Private access modifiers have the highest degree of protection. Private members are not accessible from any other class. They can not be accessed even from any subclass or any class from the same package. The methods or data members declare as private are accessible only within the class in which they are declared. Classes or interface cannot be declared as private.

3) Public Access Modifier: Any variable and methods which is declare as public. It will be accessible to all the entire class and visible to all the classes outside the class. It can be accessed from outside package also. The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

4) Protected Access Modifier: The protected access modifier is accessible within package and outside the package but through inheritance only.

3.6 Access Modifier:

The protected access modifier can be applied on the data member, methods and constructor. It cant be applied on the class. The protected members from super class can be accessed from sub class and any other class from the same package. But it can not be accessed from any class from another package.

	Default	Private	Public	Protected
Same class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non subclass	No	No	Yes	No

3.7 Inner Classes:

It is possible to define a class within another class; such classes are known as inner or nested classes. The scope of a inner class is bounded by the scope of enclosing class. Thus, if class B is defined within class A, then class B is known to class A, but not outside of class A. A inner class has access to the members, including private members of the class in which it is nested. However, the enclosing class does not have access to the members of the inner class.

It is important to realize that class inner is known only within the scope of class outer. The Java compiler generates an error massage if any code outside of class outer attempts to instantiate class inner.

3.7 Inner Classes:

```
import java.io.*;
class abc
{
        int a = 100;
        void test()
        {
                mno obj2 = new mno();
                obj2.display();
        }

        class mno
        {
                void display()
                {
                        System.out.println("Display outer a:"+a);
                }
        }
}
```

```
class innerdemo
{
        public static void main(String args[])
        {
                abc obj1 = new abc();
                obj1.test();
        }
}
```

Output:

```
Display outer a:100
```