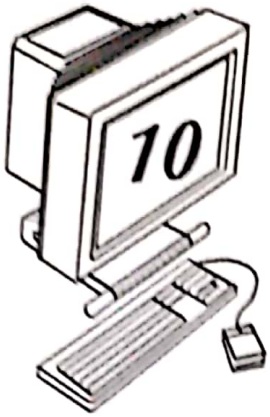

Packages

10.1 Introduction

A class is a single entity that contains the related members (data and methods) of same type of objects. If we write all the related classes, abstract classes individually, these classes will be in scattered format. Later if we want to use only these classes as a bundle of related classes, we can't use them easily.

Java environment provides a powerful means of grouping related classes and interfaces together in a single unit called **packages.** Java packages provide a convenient mechanism for managing a large group of classes and interfaces.

Classes defined within a package must be accessed through their package name. Therefore, a package provides a means to name a collection or set of classes. Each class name must be unique within a given package.

But, two classes residing in two different packages can have the same name.

Def. :

A java package is a group of similar types of classes, interfaces and sub-packages.

2 Advantages of Java Package

Advantage of packages are :

1. *Java package is used to categorize the classes and interfaces so that they can be easily maintained.*

2. The classes contained in the packages can be reused.
3. Packages reduce problems with conflicts in names. It allows us to hide classes so that conflicts can be avoided.
4. Packages allow us to protect classes, variables and methods in larger ways than on a class-by-class basis.

3 Types of Packages

There are two types of packages in Java.

1. Java API packages
2. User defined packages

3.1 Java API Packages (or Built-in packages)

Java API packages mean the built in packages of Java. Java API (Application Programming Interface) provides a large number of classes grouped into different packages according to functionality. We can use these package classes in any of the Java program by importing them at the top of the program. The Java API is stored in packages. At the top of the package hierarchy is **Java**. Descending from Java are several subpackages as shown in figure 10.1.

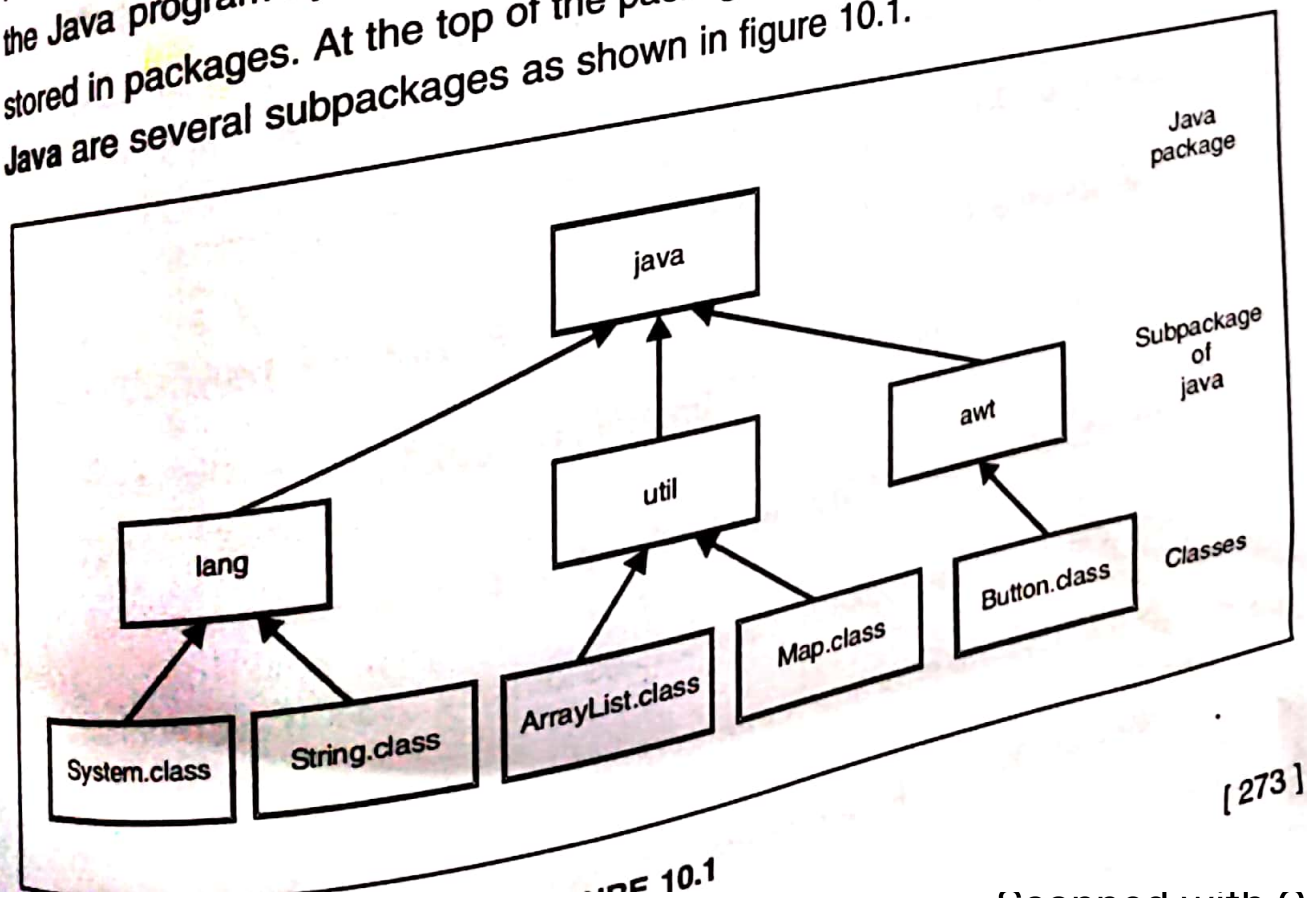

FIGURE 10.1

We have been using **java.lang** ever since the beginning of this book. It contains the **System** class, which we have been using while performing output using **println()**. The **java.lang** package is distinct and unique since it gets **automatically imported** into every Java program. However, other packages need to be explicitly imported into Java programs by using import statement. Syntax is :

```
Import packagename.classname; // import specify class only

or

import packagename.*;          //import all classes of the package
```

The first statement allows the specified class in the specified package to be imported.

The second statement imports every class contained in the specified package.

The **import statements** must appear at the top of the file, before any class declarations.

Example 10.1

The statement

import java.util.*;

imports all classes of **java.util** package.

Any class of this package can now be directly used in the program. There is no need to use the package name to qualify the class.

Most commonly used packages and their classes are given in table 10.1.

TABLE 10.1

Standard Package	Description
java.lang	Stores a large number of general purpose classes such as **System** class, **String** class and **Math** class.
java.io	Stores the Input/Output classes. For example, **DataInputStream** class.
java.util	Stores language utility classes such as vectors, date, has tables etc. For example, **Scanner** class.
java.net	Stores the classes which support networking.
java.awt	Stores classes which support the Abstract Window Toolkit. It contains classes for implementing graphical user interface.
java.applet	Stores classes for creating and implementing java applets.

3.2 User Defined Packages

Packages that are created and implemented by the user are known as **user defined packages**. Steps are :

1. Create a package using the following syntax :

package packagename;

Here

package is the keyword.

packagename is the name of package.

This statement must be first statement of the java program (except comments and white spaces.)

2. Create a directory of package name i.e. name of directory must be same as name of package.

3. Now, define the class (or classes) of the package and declare all classes as **public** (Note, a package can contain non-public classes but these classes become **hidden classes** of the package).

4. Store each class as **classname.java** in the package directory.

5. Compile the file. This creates **.class** file in the package directory.

6. Finally, use the public package members or classes outside the package. There are three ways to access public package members from outside the package.

   (i) **import package. * ;** // import an entire package.

   (ii) **import package.classname;** // import the package specify class

   (iii) **refer to the member by fully qualified name.**

Example 10.2

One example of fully qualified name is

```
double result = java . lang . Math . sqrt (x);
                package   class   method
                name      name    name
```

Note : **(Naming Conventions)**

(a) Generally, package names are written in lowercase letters.

(b) Class names and interface names are written in uppercase letters.

(c) Method names of a class are written in lowercase letters.

10.4 Defining A Package

To create a package we have to include a package command as the first statement in a Java source file. The classes declared after the statement in that file will belong to the specified package. **Syntax is**

```
package  packagename ;
```

Here, **packagename** is the name of the package and **package** is the keyword.

Example 10.3

The statement

```
package  mypkg ;
```

create a package called **mypkg**.

We can also create a hierarchy of packages, i.e. package within a package called **subpackage** for further classification and categorization of classes and interfaces.

Syntax is :

```
package mypkg.mypkg2.mypkg3;
```

Here, **mypkg2** is a subpackage in package **mypkg**, **mypkg3** is a subpackage in subpackage **mypkg2.**

Example 10.4

The statement

```
package pck1.pck2.pck3;
```

creates a hierarchy of packages. The **pck2** is a subpackage in package **pck1** and **pck3** is a subpackage in subpackage **pck2**. To refer to a class of a nested package we can use a fully qualified name with all the containing package names prefixing the class name as shown in figure 10.2.

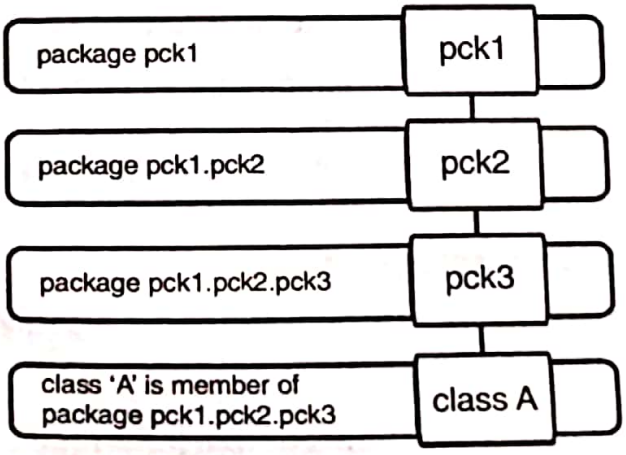

FIGURE 10.2 [Package Hierarchy]

In the above package hierarchy **class A** is identified by the name **pck1.pck2.pck3.A**. This is called the fully qualified type name of the class.

10.5 Adding a Class to package

We can add a class to a package using the following syntax.

```
package   packagename;
public class classname
{
   ...
            // body of class
   ...
}
```

Example 10.5

Step1 : Write the following code

```
package   mypack;
public class Sample
{
        public static void main (String args[ ])
        {
                System.out.println("Welcome to Package");
        }
}
```

Step 2 : Save this file as **Sample.Java**

Step 3 : Compile java package using following syntax :

Javac -d directory javafilename

for example

Javac -d . Sample.java

The -d switch specifies the destination where to put the generated class file. We can use any directory name like / home (in case of Linux), d:/abc (in case of windows) etc. If we want to keep the package within the same directory, you can use . (dot).

Step 4 : **Run java package program**

We need to use fully qualified name e.g. **mypack.Simple** to run the class. Command to run java package is :

java mypack.Sample

Output :

Welcome to package

In Java packages, every java source file contains at least one class declared as public. it can also contain non-public classes but these classes become **hidden classes** of the package. The name of the source file should be same as the name of the public class with **.java** extension.

Example 10.6

```
package pck1;

class Student
{
    int rollno;
    String name;

    public Student(int rno, String sname)
    {
        rollno = rno;
        name = sname;
    }

    public void showDetails()
    {
        System.out.println("Roll No. is " + rollno);
        System.out.println("Name is " + name);
    }
}

public class DemoPackage
{
public static void main(String ar[])
{
    Student st[]=new Student[2];
    st[0] = new Student(1001,"Amit");
    st[1] = new Student(1002,"Shivank");

    st[0].showDetails();
    st[1].showDetails();

}
}
```

Create a directory named as **pack1** and Save the program as **DemoPackage.java** in this directory.

Compile the code with the command on the command prompt.

javac –d . DemoPackage.java

To run write the command given below :

java pck1.DemoPackage

Output :

Roll No. is 1001

Name is Amit

Roll No. is 1002

Name is Shivank

We can add number of classes in a package. These classes are called as the member of the package.

Example 10.7

In this example, first create a package **areapkg** and add two classes named **Rectarea** and **Circlearea** to it.

```
// First source File
package areapkg;
public class Rectarea
{
        public double area (double l , double b)
        {
                return l * b;
        }
}
```

Note :

Create a directory **areapkg** and save the file **Rectarea.java** in it and compile it.

areapkg (Package)

Rectarea (class) Circlearea (class)

```
// Second source File

package areapkg;

public class Circlearea

{
        public double area (double r)

        {
                return 3.14 * r * r ;

        }

}
```

Note :

Save this file as **Circlearea.java** and store it in the directory **areapkg** and compile it.

Similarly, we can add any number of classes in a package.

10.6 Using Package Members

These are three ways to access the package members from outside the package:

(a) import package.* ;

(b) import package.classname;

(c) fully qualified name

(a) Using package.*

If we use package.* then all the classes and interfaces of this package will be accessible but not subpackages. The import keyword is used to make the classes and interface of another package accessible to the current package.

Example 10.8

Let us use the classes of **areapkg** package discussed earlier.

```
import areapkg.*;
class computarea
{
    public static void main(String args [ ])
    {
        // Creating object of Circlearea class
        Circlearea obj = new Circlearea( );
        double carea = obj.area(3.5);
        System.out.println("Area of circle is " + carea);

        // Creating object of Rectarea class
        Rectarea obj1 = new Rectarea();
        double rarea = obj1.area(4.5, 8.5);
        System.out.println("Area of Rectangle is " + rarea);
    }
}
```

Output :

```
Area of circle is 38.465
Area of Rectangle is 38.25
```

Note :

Save the file Named as **computarea.java** outside the directory named **areapkg.** Finally, Compile and Run it.

(b) Using packagename.classname

If you import package.classname then only declared class of this package will be accessible.

Example 10.9

```
import areapkg.Rectarea;
import areapkg.Circlearea;

class computarea1
{
    public static void main(String args [ ])
    {
            // Creating object of Circlearea class
            Circlearea obj = new Circlearea();
            double carea = obj.area(3.5);
            System.out.println("Area of circle is " + carea);

            // Creating object of Rectarea class
            Rectarea obj1 = new Rectarea();
            double rarea = obj1.area(4.5, 8.5);
            System.out.println("Area of Rectangle is " + rarea);
    }
}
```

Output :

Area of circle is 38.465

Area of Rectangle is 38.25

Note :

Save the file Named as **computarea1** outside.java the directory named **areapkg.** Finally, Compile and Run it.

(c) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

Example 10.10

```
class computarea2
{
    public static void main(String args [ ])
    {
        // Creating object of Circlearea class
        areapkg.Circlearea obj = new areapkg.Circlearea();
        double carea = obj.area(3.5);
        System.out.println("Area of circle is " + carea);

        // Creating object of Rectarea class
        areapkg.Rectarea obj1 = new areapkg.Rectarea();
        double rarea = obj1.area(4.5, 8.5);
        System.out.println("Area of Rectangle is " + rarea);
    }
}
```

Output :

```
Area of circle is 38.465
Area of Rectangle is 38.25
```

Note :

Save the file Named as **computarea2.java** outside the directory named **areapkg.** Finally, Compile and Run it.