UNIT4

Packages: Putting Classes Together

Introduction

Packages are java's way of grouping classes and interfaces together.

Benefits of using packages:

1. The classes contained in the packages of other program can be easily reused.
2. In packages, two classes in two different packages can have the same name.
3. Packages provide a way to hide classes thus preventing other programs/packages from accessing classes.
4. Packages also provide a way for separating design from coding.

JAVA API Packages

1. Java API provides a large number of classes grouped together into different packages according to functionality.
2. Below figure shows the functional breakdown of packages that are frequently used with the Java API.

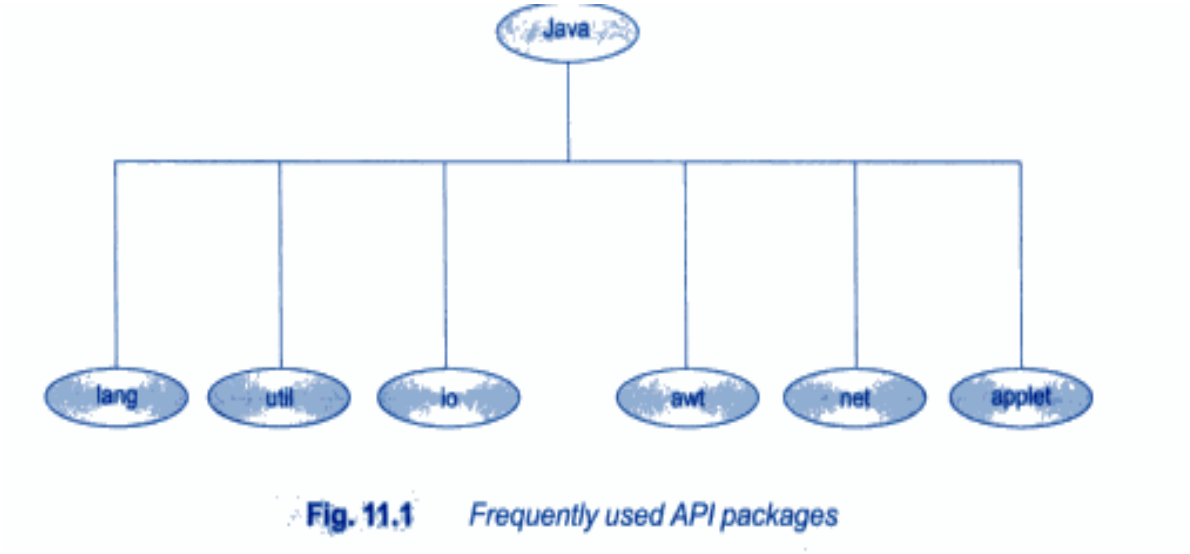


Fig. 11.1 *Frequently used API packages*

Table 11.1 Java System Packages and Their Classes

Package name	*Contents*
java.lang	Language support classes. These are classes that Java compiler itself uses and therefore they are automatically imported. They include classes for primitive types, strings, math functions, threads and exceptions.
java.util	Language utility classes such as vectors, hash tables, random numbers, date, etc.
java.io	Input/output support classes. They provide facilities for the input and output of data.
java.awt	Set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
java.net	Classes for networking. They include classes for communicating with local computers as well as with internet servers.
java.applet	Classes for creating and implementing applets.

Using System Packages

1. The packages are organized in a hierarchical structure as illustrated in figure below.
2. This shows that the package named **java** contains the package **awt**, which in turn contains various classes required for implementing graphical user interface.

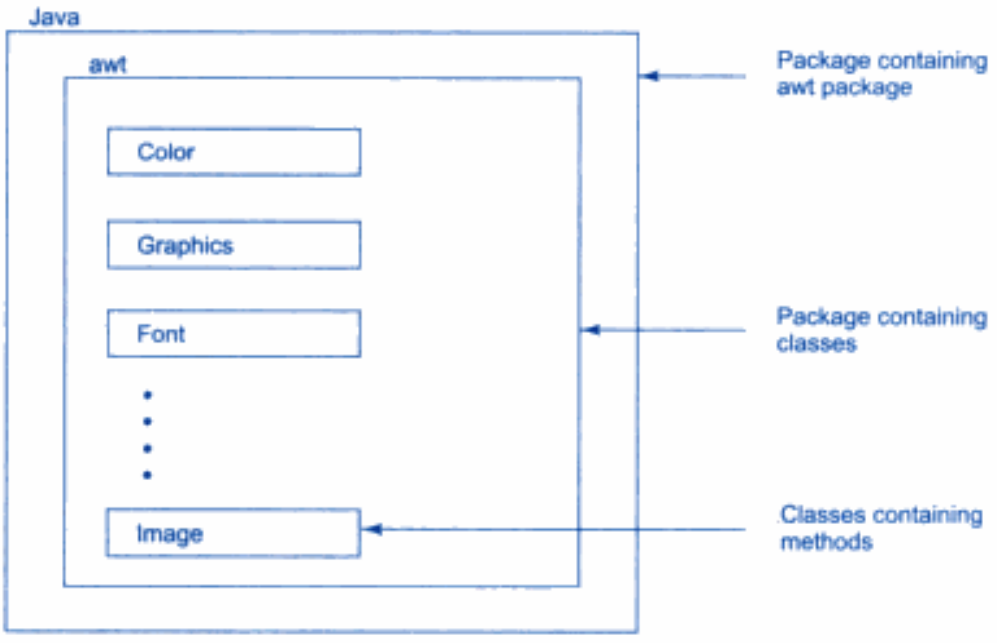


Fig. 11.2 *Hierarchical representation of java.awt package*

3. There are two ways accessing the classes stored in a package. The first approach is to use fully qualified class name of the class that we want to use

   Ex. If we want to refer to the class Color in the awt package then do as follows.

   java.awt.Color

4. Secondly in many situations we might use class in number of places in a program as follows.

   Syntax: import packagename.Classname; Ex:import java.awt.Color;

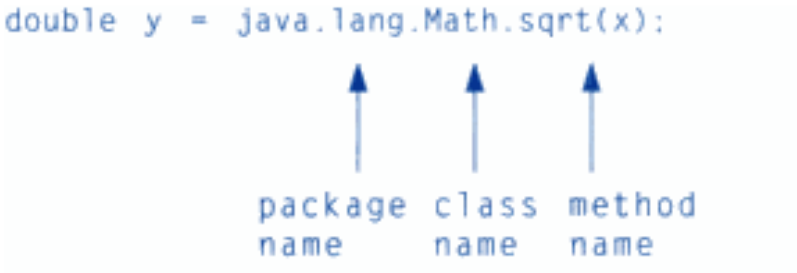
   or or

   import packagename.*; import java.awt.*;

5. These import statements must appear at the top of the file before any class declaration, import is a keyword.
6. Note that awt is a package within the packages java and the hierarchy is represented by separating levels with dots.

Naming Conventions

1. Packages can be named using standard java naming rules. All packages begin with lowercase letters and class name begin with an uppercase letter.
2. Ex.



3. This statement uses a fully qualifies class name **Math** to invoke the method **sqrt().**

Creating Packages

1. Package can be created using a keyword package followed by a package name. this must be the first statement in a java source file
2. Ex. package p1;

```
public class A
{
        ……
 }
```

3. Here the package name is p1, the class A is now considered as part of the package p1.
4. This would be saved with a file named A.java and located in a directory named p1. When the source file is compiled java will create a A**.class** file and store it in the directory p1

Creating our own packages involves the following steps.

1. Declare the package at the beginning of a file using the form

   package packagename;
2. Define the class that is to be put in the package and declare it public.
3. Create a subdirectory under the directory where main source files are stored.
4. Store the listing as classname.java file in the subdirectory created.
5. Compile the file. This creates **.class** file in the subdirectory.

Accessing a Package

1. The import statement can be used to access the list of classes from a particular package. The general form of the import statement is as follows.

   import package1.classname;

   here package1 is the name of the package and classname specifies the name of the class.
2. The import statement must end with the semicolon.
3. The import statement should appear before any class definitions in a source file.
4. The another approach of accessing package is as follows.

   import package1.*;

   here (*) indicates that we can access all the classes contained in the above package.

Using a Package

Consider some simple programs will uses classes from other packages. The listing below shows package named package1 containing class ClassA.

```
package package1;

public class ClassA
{
    public void displayA()
    {
        System.out.println("Class A");
    }
}
```

This above file should be named ClassA.java and stored in subdirectory package1.

Now consider the code below:

```
import package1.ClassA;

class test1
{
    public static void main(String args[])
    {
        ClassA a1=new ClassA();
        a1.displayA();
    }
}
```

The above shows the simple program that imports the ClassA from the package package1.

Now consider another Package named package2 containing again a single class as shown below:

```
package package2;

public class ClassB
{
    public int m=10;
    public void displayB()
    {
        System.out.println("Class B");
        System.out.println("m = "+m);
```

```
    }
}
```

The program shown below uses classes contained in both the packages and therefore it imports both the packages.

```
import package1.ClassA;
import package2.*;
class test2
{
    public static void main(String args[])
    {
        ClassA a1=new ClassA();
        ClassB b1=new ClassB();
        a1.displayA();
        b1.displayB();
    }
}
```

When we import multiple packages it is likely that two or more packages contain classes with similar names. For example

```
package p1;
public class Teacher
{……...}
protected class Student
{……}
```

```
package p2;
public class Course
{……….....}
protected class Student
{……….....}
```

Since both the packages contain the class Student compiler cannot understand which one to use and therefore generates an error.

In such cases we have to be more explicit about which one we intend to use For Ex..

```
import p1.*;
import p2.*;
p1.Student s1;
p2.Student s2;
```

Subclassing an imported class

It is possible to subclass a class that has been imported from another package. This is illustrated in below program.

```
import package2.ClassB;
class ClassC extends ClassB
{
        int n=20;
        void display()
        {
                System.out.println("Class C");
                System.out.println("m =" + m);
```

```
            System.out.println("n =" + n);
        }
}
class Test3
{
        public static void main(String args[])
        {
            ClassC c1=new ClassC();
            c1.displayB();
            c1.displayC();
        }
}
```

while using packages and inheritance in a program we should be aware of the visibility restrictions of various access specifiers. The access protection details is given below in the table.

Table 11.2 Access Protection

Access location \ Access modifier →	*public*	*protected*	*friendly (default)*	*private protected*	*private*
Same class	Yes	Yes	Yes	Yes	Yes
Subclass in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Subclass in other packages	Yes	Yes	No	Yes	No
Non-subclasses in other packages	Yes	No	No	No	No

Adding a Class to a Package

1. it is simple to add a class to an existing package. Consider the following package.

```
package p1;
public class A
{
        //body of the class A
}
```

2. The package p1 contains one public class by name A. Suppose we want to add another class B to this package. This can be done as follows:
   a. Define the class and make it public.

      Place this package statement → package p1; before the class definition as follows

```
package p1;
public class B
{
        // body of the class B
}
```

   b. store this as B.java file under the directory p1.
   c. Compile B.java file. This will create B.class file and place it in the directory p1.

Java source file can have only one class declared as public, we cannot put two or more public classes together in a **.java** file. This is because of the restriction that file name should be same as the name of the public class with **.java** extension.

If we want to have more than one public class in a single package then follow the following steps.

a. Decide the name of the package.
b. Create a subdirectory with this name under the directory where main source files are stored.
c. Create classes that are to be placed in the package in separate source files and declare the package statement.

```
package packagename;
```

This may look like this.

```
//source file in redlabel sub-directory is Tea.java
package redlabel;
public class Tea
{
    // body of class Tea
}
//source file in redlabel sub-directory is Coffee.java
package redlabel;
public class Coffee
{
    // body of class Coffee
}
//source file in redlabel sub-directory is Sugar.java
package redlabel;
public class Sugar
{
    // body of class Sugar
}
```

d. Compile each source file to get respective **.class** files.

Hiding Classes

1. We may like to hide classes from accessing from outside of the package.
2. Such classes can be declared "not public".

Ex..
```
package p1;
public class X                  //public class, available outside
{
        //body of X
}
class Y                         //not public class, hidden
{
        //body of Y
}
```

3. Here, the class Y which is not declared public is hidden outside of the package p1.This class can be seen and used only by other classes in the same package.

Static Import

1. Static import suggests that an import statement contains static keyword for importing static members of the class
2. This feature eliminates the need of qualifying a static member with the class name.
3. We can use the static import statement to import static members from classes and use them without qualifying the class name.
4. The syntax for using static import feature is:

```
import static package_name.subpackage_name.class_name.*;
```

For example(with static import)

```
import static java.lang.Math.*;
double y = pow(x,y);          //No qualifying classname as prefix
double z = sqrt(x);           //and are valid with
int top = max(p,q);           //static import statement
```

For example(without static import)

```
import java.lang.Math.*;
double y = Math.pow(x,y);     //Qualifying names are required to be
double z = Math.sqrt(x);      //prefixed with .Math
int top = Math.max(p,q);      //everytime while accessing methods
```

//Program to demonstrate the use of Static import

```
import static java.lang.Math.*;
public class mathop
{
    double r = sqrt(20);
    public void circle()
    {
        double area = PI * r * r;
        System.out.println("The area of circle is : " +area);
    }
```

```
public static void main(String args[ ])
        {
                mathop m1 = new mathop();
                m1.circle();
        }
}
```

//Write a package program to demonstrate basic arithmetic operators.

```
        package arithmetic;
        public class math
        {
                public int add(int x,int y)
                {
                        return(x+y);
                }
                public int sub(int x,int y)
                {
                        return(x-y);
                }
                public int mul(int x,int y)
                {
                        return(x*y);
                }
```

```
    public double div(double x,double y)
    {
        return(x/y);
    }
    public double mod(double x,double y)
    {
        return(x%y);
    }
}


import arithmetic.math;
class test
{
    public static void main(String args[])
    {
        math m=new math();
        System.out.println (m.add(4, 2));
        System.out.println (m.sub (4, 2));
        System.out.println (m.mul (4, 2));
        System.out.println (m.div (4, 2));
        System.out.println (m.mod (4, 2));
    }
}
```

//Write a program to use inbuilt packages to calculate the squareroot.

```
import java.util.Scanner.*;
import static java.lang.Math.*;
public class square_root
{
    public static void main(String args[])
    {
        double square, n;
        Scanner s=new Scanner (System. in);
        System.out.println ("Enter a number to calculate squareroot");
        n=s.nextDouble ();
        square=sqrt(n);
      System.out.println ("The squareroot of the given number is="+square);
    }
}
```