UNIT - II

Packages :-

By using packages we can use classes from other programs without physically copying them into the program under development

Package is a way of achieving the reusability in Java.

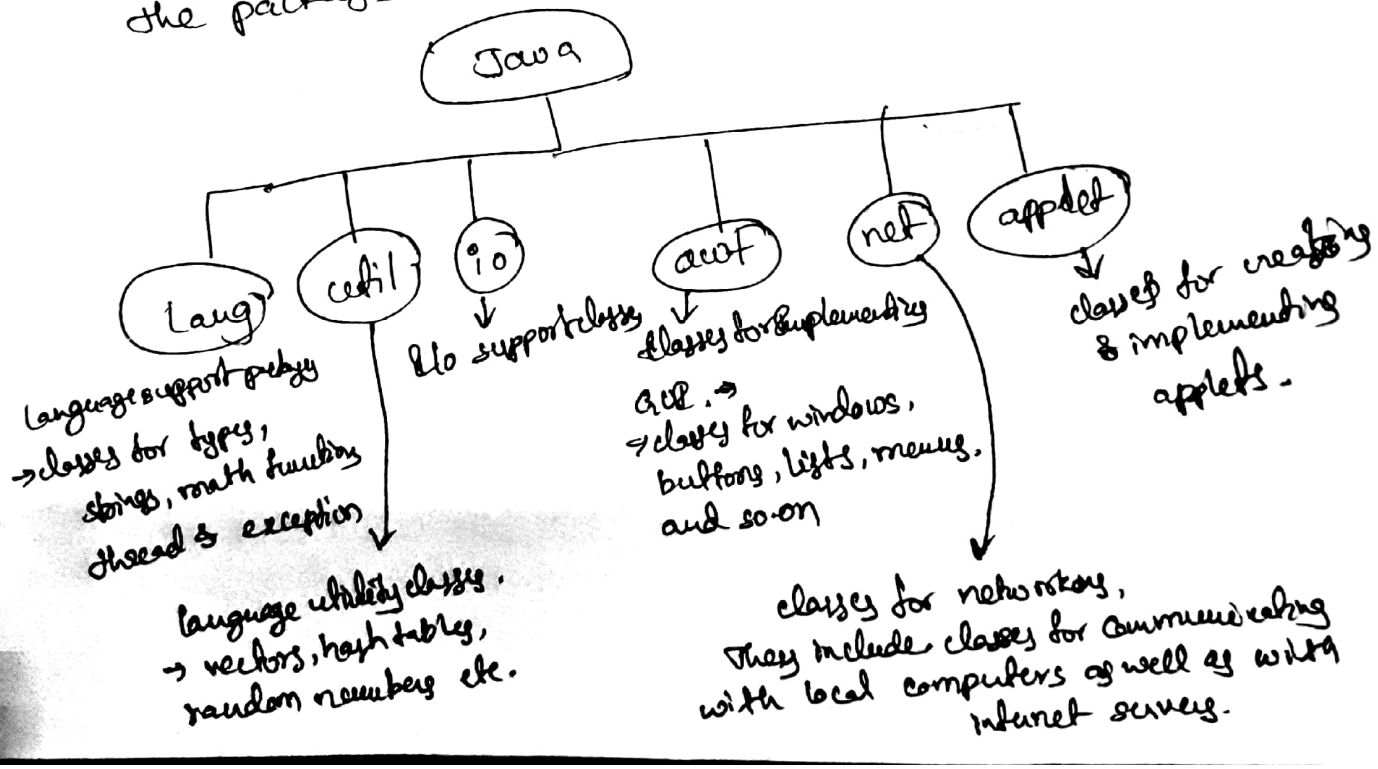
→ Packages are Java's way of grouping a variety of classes / or interfaces together. The grouping is done according to functionality. In fact, packages act as "containers" for classes.

→ Packages are classified into 2 types

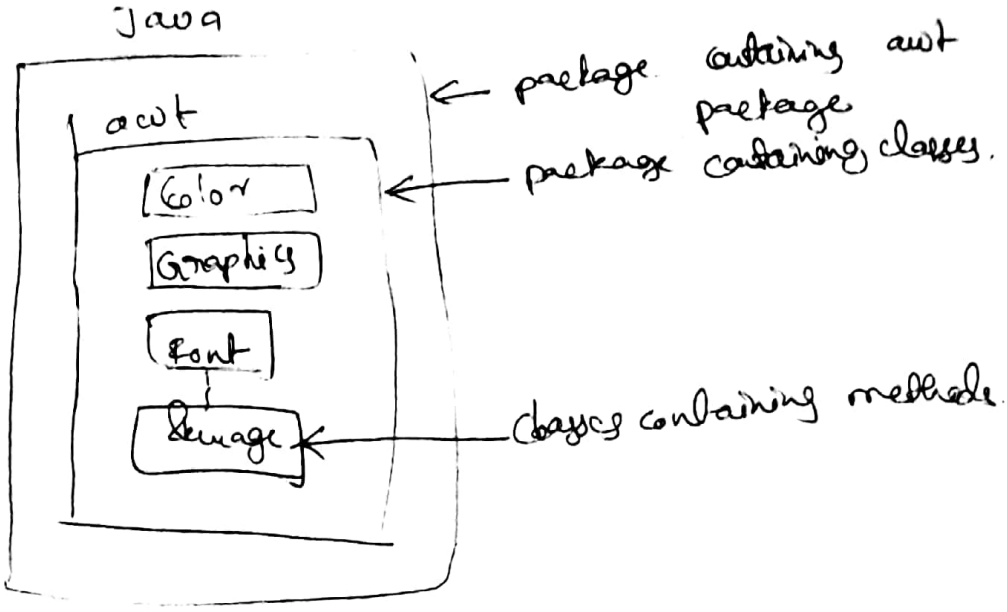
1) Java API packages
2) user defined packages

1) Java API package

Java API provides a large no. of classes grouped into different packages according to functionality. Most of the times we use the packages available with the Java API.



Ex:-



Hierarchical representation of java.awt. package

→ There are two ways of accessing the classes stored in a package

1) fully qualified class name

Ex:- java.awt.Color

```
Double y=java.lang.Math.sqrt(2);
```

pack name ↑ class name ↓ method name ↓

→ Here awt is a package within the package java and the hierarchy is represented by seperating the levels with dots. This approach is perhaps the best & easiest one if we need to access the class only once when we need not have to access any other classes of the package

→ import packagename.classname;

or

import packagename.*;

2) The second statement imports every class contained in the specified package (shortcut approach)

ex:- import java.awt.*;
import java.lang.*;

Naming Conventions in Packages:-

Packages begin with lower case letters.
This makes it easy for users to distinguish package names from class names when looking at an explicit reference to a class.

ex:- double y= java.lang.Math.sqrt(x);

Creating User defined Packages:

Syntax:- package firstPackage; // package declaration
public class FirstClass
{
-- (body of class)
}

Ex:- package packagename;
package firstpack.SecondPackage;
(means firstpackage/secondpackage)

Note:- A java package file can have more than one class def's. In such cases, only one of the classes may be declared public and that class name with .java extention is the source file name. When a source file with more than one class definition is compiled, java creates independent .class files for those classes..

Accessing a Package:-

a Java system package can be accessed either using a fully qualified class name or using a shortcut approach through the import statement

general syntax

1) import package1[.package2][.package3].classname;

top level package package inside ... soon.
package1

@ import firstPackage.secondPackage.MyClass;

2) import packagename.*;

import firstPackage.*;

Access Protection:-

	private	No modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same Package Subclass	No	Yes	Yes	Yes
Same Package non subclass	No	Yes	Yes	Yes
Different Package subclass	No	No	No	Yes
Different Package non subclass	No	No	No	Yes

Access Protection :-

create two packages - P1 and P2

→ the source for first package (P1) defines three classes
↳ Protection, Derived, SamePackage.

→ In Protection → define n variable with default protection.
n_pri is private.
n_pro is protected
n_pub is public

→ Derived is a subclass of Protection in the same package P1.
↳ It grant access to every variable in Protection
except for n_pri.

→ SamePackage is not a subclass of Protection.
↳ It also access all variables except n_pri

→ the source for second package P2 defines 2 classes

→ The Protection2 is subclass of P1.Protection
↳ This grants access to all of P1.Protection's variables except for n_pri and n.

→ The OtherPackage has access to only one variable, n_pub, which was declared as public.

```
package P1;
public class Protection
{   int n=1;
    private int n_pri=2;
    protected int n_pro=3;
    public int n_pub=4;
    public Protection()
    {  System.out.println("base constructor");
       System.out.println("n= "+n);
       System.out.println("n_pri= "+n_pri);
       System.out.println("n_pub= "+n_pub);
       System.out.println("n_pro= "+n_pro);
    }
}
```

Protection.java

>javac Protection.java

only compilation

Derived.java

```
package P1;
class Derived extends Protection
{
    Derived()
    { System.out.println("Derived constructor");
      System.out.println("n= "+n);
      //class only
      //System.out.println("n_pri= "+n_pri);
      System.out.println("n_pro= "+n_pro);
      System.out.println("n_pub= "+n_pub);
    }
}
```

SamePackage.java

```
class SamePackage
{
    SamePackage()
    { Protection p=new Protection();
      System.out.println("same package constructor");
      System.out.println("n= "+p.n);
      //class only
      //System.out.println("n_pri= "+p.n_pri);
      System.out.println("n_pro= "+p.n_pro);
      System.out.println("n_pub= "+p.n_pub);
    }
}
```

test file for package P1

```
package P1;
public class Demo
{ public static void main(String args[])
    { Protection ob1=new Protection();
      Derived ob2=new Derived();
      SamePackage ob3=new SamePackage();
    }
}
```

Protection2.java

```
package p2;
class Protection2 extends p1.Protection
{
    Protection2()
    {
        System.out.println("derived other package constructor");
        //class or package only
        //system.out.println("n="+n);
        //class only
        //system.out.println("n_pri="+n_pri);
        System.out.println("n_pro="+n_pro);
        System.out.println("n_pub="+n_pub);
    }
}
```

OtherPackage.java

```
package p2;
class OtherPackage
{   OtherPackage()
    {   p1.Protection p=new p1.Protection();
        System.out.println("other package constructor");
        //class or package only
        //system.out.println("other package constructor");
        //class only or package only
        //system.out.println("n="+p.n);
        //class only
        //system.out.println("n_pri="+n.pri);
        //class, subclass or package only
        //System.out.println("n_pro="+p.n_pro);
        System.out.println("n_pub="+p.n_pub);
    }
}
```

test file for p2

```
package p2;
public class Demo
{ public static void main(String args[])
  { Protection2 obj1 = new Protection2();
    OtherPackage obj2 = new OtherPackage();
  }
}
```

Importing Packages :-

→ import stmts occur immediately following the package stmt and before any class definitions.

general form of the import stmt

```
import pkg[.pkg2].(classname/*);
```

pkg — top level package
pkg2 — subordinate package inside the outer package
(classname/*) → indicates that java compiler should import the entire package

ex:-
```
import java.util.*;
class MyDate extends Date
{ ...
}
```

→ same example without import statement looks like.

```
class MyDate extends java.util.Date
{ ...
}
```

Note:- When a package is imported, only those items within the package declared as public will be available to non-subclasses in the importing code.

Example program

→ Balance.java

```
package Mypack;
public class Balance
{
    String name;
    double bal;
    public Balance(String n, double b)
    {   name=n;
        bal=b;
    }
    public void show()
    {   if(bal<0)
        System.out.println("-->");
        System.out.println(name+": $"+bal);
    }
}
```

>javac -d. Balance.java
>cd mypack
>java Balance
or
>java mypack.Balance

→ TestBalance.java

Run this TestBalance imports mypack and is then able to make use of the Balance class.

```
→ import mypack.*;
class TestBalance
{   public static void main(String args[])
    {   Balance test=new Balance("ABC", 123.30);
        test.show();
    }
}
```

>javac TestBalance.java
>java TestBalance

O/P: ABC: $123.30

Note:- As an experiment, remove the public specifier from the Balance class and then try compiling TestBalance. As explained, errors will result.

A short Package Example:-

```
package Mypack;
class Balance
{ String name;
  double bal;
  Balance(String n, double b)
  { name=n;
    bal=b;
  }
  void show()
  { if(bal<0)
    System.out.println("-->");
    System.out.println(name+": $"+bal);
  }
}
```

```
class AccountBalance
{ public static void main(String args[])
  { Balance current[]=new Balance[3];
    current[0]=new Balance("Abc",
                           123.23);
    current[1]=new Balance("Def",
                           157.02);
    current[2]=new Balance
                ("Ghi", -12.33);
    for(int i=0; i<3; i++)
      current[i].show();
  }
}
```

o/p: 1) javac -d. AccountBalance.java
     java Mypack.AccountBalance

2) way is create director mypack and put AccountBalance.class in that directory and compile the file, make sure that .class file is also in the mypack and execute the file using the following command.

java Mypack.AccountBalance.

o/p: Abc: $123.23
Def: $157.02
-->

classpath is actually an environment variable in Java, and tells Java application and JVM where to find the libraries of classes. They include any that you have developed on your own.

→ An environment variable is a global system variable, accessible by the computer's operating system (eg. windows). Other variable's include COMPUTERNAME, USERNAME.

→ In Java, CLASSPATH holds the list of Java class file directories, and the JAR file, which is Java's delivered class library file.

two types of path setting

1) Temporary classpath setting

In command prompt C:> set CLASSPATH = C:\Java

2) Permanent classpath setting

1) select start
   ↳ control panel (select
      ↳ System & security (double click)
         ↳ select Advanced tab

2) Click Environment variables.
In the section System variables, find the PATH environment variable and select it. Click Edit.
If the PATH environment variable doesn't exist, click New.

set variable : PATH
value : C:\Users\programfiles\Java\JDK 1.8\bin

5) On the Edit System variable (or New System variable) window, specify the value of PATH environment variable, C
close all remaining windows by clicking Ok.