CHAPTER - 6

PACKAGES

6. Packages

Package is a collection of related classes. Each class defines number of methods. Java packages are classified into 2 types.

1. Java API (Application Programming Interface) packages (or) Predefined packages (or) Built-in packages: These packages are defined by the system. Some of the example for system defined packages are java.lang, java.util, java.io, java.awt, java.applet etc., the JDK1.3 contains nearly 60 predefined packages.
2. User defined packages: These packages are defined by the user.

6.1 Defining a Package:

To define a package, place "package" command as the first statement in the Java source file. So, that any class declared within that file will belong to the specified package. The syntax of package creation is as follows.

Syntax: package pack-name;

where pack-name is the name of the package.

Example:

```
package mypack;
public class number
{
public void add (int a,int b)
{
System.out.println("Sum="+(a+b));
}
}
```

- The class that is defined in the package must be start with the public access modifier. So, that it can be accessible by any another of them. If it is not public, it is possible to access only in that package.
- Java uses file system directories to store packages. We save the program with number.java and compile the package is as javac –d . number.java

  Due to this compilation mypack directory is automatically created and .class file is stored in that directory.
- Package creation has completed. The package information is now including in our actual program by means of "import" statement. "import" is a keyword that links the package with our program. It is placed before the class definitions.

```
import mypack.*;
or
import mypack.number;
```

Example 1:

```
import mypack.number;
class pack
{
    public static void main(String[] args)
    {
        number obj=new number();
        obj.add(3,4);
    }
}
```

Example 2:

```
package math1;
public class Mcal
{
    public void square(int x)
    {
    System.out.println("Square of "+x+" is"+(x*x));
    }
    public void cube(int x)
```

```
    {
    System.out.println("cube "+x+" is"+(x*x*x));
    }
}

import math1.Mcal;
class calc
{
    public static void main(String[] args)
    {
        Mcal obj=new Mcal();
        obj.square(3);
    ____ obj.cube(4);
    }
}
```

Example 3:

```
package p6;
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(name+"---"+bal);
}
}
}
```

```
import p6.Balance;
class Account
{
      public static void main(String[] args)
      {
      Balance c[]=new Balance[3];
      c[0]=new Balance("John",123.33);
      c[1]=new Balance("Arun",157.02);
      c[2]=new Balance("Siri",-12.33);
      for(int i=0;i<3;i++)
      c[i].show();
      }
}
```

Note: 1. Java source file contains only one public class and any number of non-pubic classes. The filename should be same as the name of the public class with .java extension.

2. Package contain any number of public and default classes.

```
Example:package p1;
public class x
{
 // body of x        ⟶  this class is accessible
}
class y
{
// body of y         ⟶  not accessible due to non-use
} of public access modifier
```

Let we want to access all the classes of information then the classes are stored in different files with the same package name.

Example:

```
package mypack;
public class number
{
public void add (int a,int b)
{
System.out.println("Sum="+(a+b));
}
```

```
}

package mypack;
public class number1
{
public void sub(int a,int b)
{
System.out.println("sub="+(a-b));
}
}
import mypack.number;
import mypack.number1;

class pack
{
    public static void main(String[] args)
    {
        number obj=new number();
        obj.add(3,4);
        number1 obj1=new number1();
        obj1.sub(6,2);

}
```

6.2. Sub Packages:

It is also possible to create sub packages for the main package.

Syntax: package pack1.pack2;

where pack1 is the main package and pack2 is the sub package.

Example:

```
package pack1;
public class x
{

    public void show()
    {
    System.out.println("Super");
    }
```

```
}
package pack1.pack2;
public class y
{
        public void display()
        {
        System.out.println("Sub");
        }
}

import pack1.x;
import pack1.pack2.y;
class che
{
        public static void main(String[] args)
        {
                x obj=new x();
                obj.show();
                y obj1=new y();
                obj1.display();
        }
}
```

6.3 Access Protection

Java provides four types Access modifiers as public, private, default and protected. Any variable declared as public, it could be accessed from anywhere. Any variable declared as private cannot be seen outside of its class. Any variable declared as default, it is visible to subclasses as well as to other classes in the same package. Any variable declared as protected, it allows an element to be seen outside of current package, but only to classes that subclass directly.

	public	private	default	protected
Same class	Yes	Yes	Yes	Yes
Same package sub class	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package sub class	Yes	No	No	Yes
Different package non-subclass	Yes	No	No	No

Example:

```
package p1;
public class protect
{
    int n=1;
    private int n_pri=2;
    protected int n_pro=3;
    public int n_pub=4;
    public protect()
    {
    System.out.println("Super constructor");
    System.out.println("default="+n);
    System.out.println("private="+n_pri);
    System.out.println("protected="+n_pro);
    System.out.println("pubic="+n_pub);
    }
}
package p1;
class derive extends protect
{
    derive()
    {
    System.out.println("Sub constructor");
    System.out.println("default="+n);
    //System.out.println("private="+n_pri);
    //since it is a private variable in package p1
    System.out.println("protected="+n_pro);
    System.out.println("pubic="+n_pub);
    }
}
import p1.protect;
class same
{
    public static void main(String[] args)
    {
        protect obj=new protect();
    }
}
```

6.4 Classpath

A class path is an environmental variable, which tells the java virtual machine and other java tools (java, javac), where to find the class libraries, including user-defined class libraries.

By default, java uses the classpath as

C:\jdk1.2.1\lib\classes.zip

A user defined class path is set for the environment as

C:\>SET CLASSPATH=%CLASSPATH%;C:\P (or)

C:\>SET PATH="jdk1.2.1\bin"

6.5. Example programs

EXAMPLE :: 1

```
class StackDemo{
  public static void main(String args[]) {
  .Stack st1= new Stack();
   Stack st2 = new Stack();
   // push some numbers onto the stack
   for(int i=0; i<10; i++) st1.push(i);
   for(int i=10; i<20; i++) st2.push(i);
   // pop those numbers off the stack
   System.out.println("Stack in st1:");
   for(int i=0; i<10; i++)
     System.out.println(st1.pop());
   System.out.println("Stack in st2:");
   for(int i=0; i<10; i++)
     System.out.println(st2.pop());
   // these statements are not legal
   // st1.tos = -2;
   // st2.stck[3] = 100;
  }
}
```

OUTPUT

```
C:\ >javac StackDemo.java
C:\ >java StackDemo
```

EXAMPLE :: 2

```
// Improved Stack class that uses the length array member.
class Stack {
 private int stk[];
 private int tos;
 // allocate and initialize stack
 Stack(int size) {
  stk = new int[size];
  tos = -1;
 }
// Push an item onto the stack
 void push(int item) {
  if(tos==stk.length-1) // use length member
   System.out.println("Stack is full.");
  else
   stk[++tos] = item;
 }
 // Pop an item from the stack
 int pop() {
  if(tos < 0) {
   System.out.println("Stack underflow.");
   return 0;
  }
  else
   return stk[tos--];
 }
}
class StackDemo2 {
 public static void main(String args[]) {
  Stack st1 = new Stack(5);
  Stack st2 = new Stack(8);
  // push some numbers onto the stack
  for(int i=0; i<5; i++) st1.push(i);
  for(int i=0; i<8; i++) st2.push(i);
  // pop those numbers off the stack
  System.out.println("Stack in mystack1:");
```

```
    for(int i=0; i<5; i++)
      System.out.println(st1.pop());
    System.out.println("Stack in mystack2:");
    for(int i=0; i<8; i++)
      System.out.println(st2.pop());
  }
}
```

C:\ >javac StackDemo2.java
C:\ >java StackDemo2

EXAMPLE :: 3

```
// Demonstrate static variables, methods, and blocks.
class UseStaticdemo {
 static int x = 3;
 static int y;

 static void moth(int z) {
  System.out.println("z = " +z);
  System.out.println("x = " + x);
  System.out.println("y = " +y);
 }
 static {
  System.out.println("Static block initialized.");
 y = x * 4;
 }
 public static void main(String args[]) {
  moth(42);
 }
}
```

OUTPUT:

C:\ >javac UseStaticdemo1.java
C:\ >java UseStaticdemo1
Z = 42
X = 3
Y = 12

EXAMPLE :: 4

```
// Using abstract methods and classes.
abstract class A {
 double x;
 double y;
 A(double a, double b) {
  x = a;
  y = b;
 }
// area is now an an abstract method
 abstract double area();
}
class B extends A {
 B(double a, double b) {
  super(a, b);
 }
 // override area for B
 double area() {
  System.out.println("Inside Area for B.");
  return x * y;
 }
}
class C extends A {
 C(double a, double b) {
  super(a, b);
 }

 // override area for right C
 double area() {
  System.out.println("Inside Area for Triangle.");
  return  (x*y)/ 2;
 }
}

class AbstractAreasDemo {
 public static void main(String args[]) {
 //  A f = new A(10, 10); // illegal now
```

```
    B r = new B(9, 5);
    C t = new C(10, 8);

    A  x;// this is OK, no object is created

    x = r;
    System.out.println("Area is " + x.area());

    x = t;
    System.out.println("Area is " + x.area());
  }
}
```

OUTPUT:

```
C:\ >javac AbstractAreasDemo.java
C:\ >java AbstractAreasDemo
Inside Area for B
Area is 45.0
Inside Area for Triangle.
Area is 40.0
```

EXAMPLE :: 5

```
// A Simple demonstration of abstract.
abstract class A {
 abstract void call1();

 // concrete methods are still allowed in abstract classes
 void call2() {
  System.out.println("This is a concrete method.");
 }
}

class B extends A {
 void call1() {
  System.out.println("B's implementation of callme.");
 }
}
```



```
class AbstractDemo {
 public static void main(String args[]) {
  B b = new B();
b.call1();
  b.call2();
 }
}
```

OUTPUT:

```
C:\ >javac AbstractDemo.java
C:\ >java AbstractDemo
B's implementation of callme.
This is a concrete method.
```

EXAMPLE :: 6

```
// Demonstrate when constructors are called.

// Create a super class.
class A {
 A() {
  System.out.println("Inside A's constructor.");
 }
}

// Create a subclass by extending class A.
class B extends A {
 B() {
  System.out.println("Inside B's constructor.");
 }
}

// Create another subclass by extending B.
class C extends B {
 C() {
  System.out.println("Inside C's constructor.");
 }
}
```

```
class CallingConsDemo {
 public static void main(String args[]) {
  C c = new C();
 }
}
```

OUTPUT:

```
C:\ >javac CallingCons.java
C:\ >java  CallingCons
Inside A's constructor.
Inside B's constructor.
Inside C's constructor.
```

EXAMPLE :: 7

```
// Method overriding. demo
class A {
 int i, j;

 A(int a, int b) {
  i = a;
  j = b;
 }

 // display i and j
 void show() {
  System.out.println("i and j: " + i + " " + j);
 }
}

class B extends A {
 int k;

 B(int a, int b, int c) {
  super(a, b);
  k = c;
 }
```



```
 // display k -- this overrides show() in A
 void show() {
  System.out.println("k: " + k);
 }
}

class OverrideDemo {
 public static void main(String args[]) {
  B b = new B(1, 2, 3);

  b.show(); // this calls show() in B
 }
}
```

OUTPUT:

```
C:\ >javac OverrideDemo.java
C:\ >java  OverrideDemo
K=3
```

EXAMPLE :: 8

```
// Methods with differing type signatures are overloaded -- not overridden.
class X {
 int i, j;

 X(int a, int b) {
  i = a;
  j = b;
 }

 // display i and j
 void show() {
  System.out.println("i and j: " + i + " " + j);
 }
}
```

```
// Create a subclass by extending class A.
class Y extends X {
 int k;
Y(int a, int b, int c) {
  super(a, b);
  k = c;
 }
 // overload show()
 void show(String msg) {
  System.out.println(msg + k);
 }
}
class Override1 {
 public static void main(String args[]) {
  Y b = new Y(1, 2, 3);
b.show("This is k: "); // this calls show() in Y
  b.show(); // this calls show() in X
 }
}
```

OUTPUT:

```
C:\ >javac OverrideDemo1.java
C:\ >java  OverrideDemo1
This is k:3
i and j:1 2
```

EXAMPLE :: 9

```
// A simple example of inheritance.
// Create a superclass.
class A {
 int i, j;
 void showij() {
  System.out.println("i and j: " + i + " " + j);
 }
}
```



```
// Create a subclass by extending class A.
class B extends A {
 int k;
 void showk() {
  System.out.println("k: " + k);
 }
 void sum() {
  System.out.println("i+j+k: " + (i+j+k));
 }
}

class SimpleInheritance {
 public static void main(String args[]) {
  A a = new A();
  B b = new B();
  // The superclass may be used by itself.
  b.i = 10;
 b.j = 20;
  System.out.println("Contents of b: ");
 b.showij();
  System.out.println();
  /* The subclass has access to all public members of
    its superclass. */
  b.i = 7;
  b.j = 8;
  b.k = 9;
  System.out.println("Contents of subOb: ");
  b.showij();
  b.showk();
  System.out.println();
  System.out.println("Sum of i, j and k in b:");
  b.sum();
 }
}
```

OUTPUT:

```
C:\>javac SingleInheritance.java
C:\>java  SingleInheritance
Contents of b:
i and j:10 20
Contents of subOb:
i and j:7 8
k: 9
Sum of i, j and k in b:
i+j+k:24
```

EXAMPLE :: 10

```
// Using super to overcome name hiding.
class A {
  int i;
}

// Create a subclass by extending class A.
class B extends A {
  int i; // this i hides the i in A
B(int a, int b) {
    super.i = a; // i in A
    i = b; // i in B
  }
 void show() {
   System.out.println("i in superclass: " + super.i);
    System.out.println("i in subclass: " + i);
  }
}
 class UseSuperDemo {
  public static void main(String args[]) {
    B b = new B(1, 2);

    b.show();
  }
 }
```


OUTPUT:

```
C:\>javac UseSuperDemo.java
C:\>java  UseSuperDemo
i in superclass : 1
i in subclass : 2
```

EXAMPLE :: 11

```
// Demonstrate static variables, methods, and blocks.
class UseStaticDemo {
 static int a = 10;
 static int b;
static void meth(int x) {
   System.out.println("x = " + x);
   System.out.println("a = " + a);
   System.out.println("b = " + b);
 }
static {
   System.out.println("Static block initialized.");
  b = a * 6;
 }
public static void main(String args[]) {
   meth(42);
 }
}
```

OUTPUT:

```
C:\>javac UseStaticDemo.java
C:\>java  UseStaticDemo
Static block initialized.
x = 42
a = 10
b = 60
```

EXAMPLE :: 12

```
// Dynamic Method Dispatch
class A {
  void callme() {
    System.out.println("Inside A's callme method");
  }
}
```

```
class B extends A {
 // override callme()
 void callme() {
  System.out.println("Inside B's callme method");
 }
}
class C extends A {
 // override callme()
 void callme() {
  System.out.println("Inside C's callme method");
 }
}
class Dispatch {
 public static void main(String args[]) {
  A aa = new A(); // object of type A
  B bb= new B(); // object of type B
  C cc = new C(); // object of type C
  A rr; // obtain a reference of type A
  rr = aa; // rr refers to an A object
  rr.callme(); // calls A's version of callme
  rr = bb; // rr refers to a B object
  rr.callme(); // calls B's version of callme
  rr = cc; // rr refers to a CC object
  rr.callme(); // calls C's version of callme
 }
}
```

OUTPUT:

```
C:\ >javac Dispatch.java
C:\ >java  Dispatch
Inside A's callme method
Inside B's callme method
Inside C's callme method
```