Packages & Inbuilt classes

Syeda Ajbina Nusrat, Lecturer, CSE, UITS

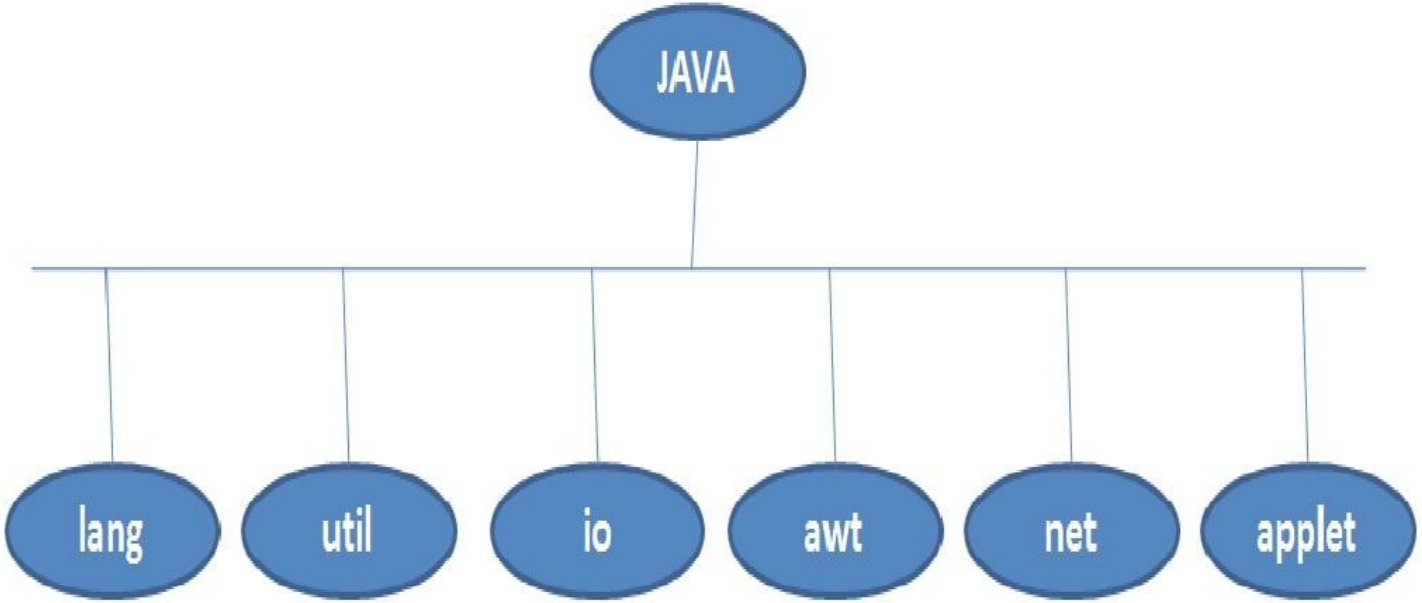
Package :

Packages are java`s way of grouping a veriety of classes and/or interfaces together.

Java API packages :

java API provides a large number of classes grouped into different packages according to functionality.

Frequently used API packages are a[illegible]nder.


JAVA
lang
util
io
awt
net
applet

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 the 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.

Syntax for import packages is as under

```
import packagename.classname;
```

or

```
import packagename.*;
```

These are known as import statements and must appear at the top of the file, before any file declaration as you can see in our few examples. Here **import** is a keyword.

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

EX :

```
import java.awt.Color;
double y =  java.lang.Math.sqrt(x);
```

Here lang is a package, Math is a class and sqrt is a method.

For create new package you can write...

```
package firstPackage;
public class Firstclass
{
        ..................
        body of class
        ..................
}
```

Vector class :

The Vector class is one of the most important in all of the Java class libraries. We cannot expand the size of a static array.

We may think of a **vector as a dynamic array** that automatically expands as more elements are added to it.

All vectors are created with some initial capacity.

When space is needed to accommodate more elements, the capacity is automatically increased. That is why vectors are commonly used in java programming.

This class provides the following

constructors:

Vector()

Vector(**int n**)

Vector(**int n, int delta**)

The first form creates a vector with an initial capacity of ten elements.

The second form creates a vector with an initial capacity of n elements.

The third form creates a vector with an initial capacity of n elements that increases by delta elements each time it needs to expand.

```java
import java.util.*;
public class Vector_Demo
{
   public static void main(String args[])
   {
      int i;
      Vector v = new Vector();
      v.addElement(new Integer(10));
      v.addElement(new Float(5.5f));
      v.addElement(new String("Hi"));
      v.addElement(new Long(2500));
      v.addElement(new Double(23.25));
      System.out.println(v);
      String s = new String("Bhagirath");
      v.insertElementAt(s,1);
      System.out.println(v);
      v.removeElementAt(2);
      System.out.println(v);
      for(i=0;i<5;i++)
      {
         System.out.println(v.elementAt(i));
      }
   }
}
```

Output :

```
Bhagirath
Hi
2500
23.25
```

The Random class allows you to generate random double, float, int, or long numbers.

This can be very helpful if you are building a simulation of a real-world system.

This class provides the following constructors.

Random()
Random(**long start**)

Here, start is a value to initialize the random number generator.

Method	Description
Double nextDouble()	Returns a random double value.
Float nextFloat()	Returns a random float value.
Double nextGaussian()	Returns a random double value. Numbers obtained from repeated calls to this method have a Gaussian distribution with a mean of 0 and a standard deviation of 1.
Int nextInt()	Returns a random int value.
Long nextLong()	Returns a random long value.
Method	Description

```
import java.util.*;
public class Random_Demo
{
   public static void main(String args[])
   {
      Random ran = new Random();
      for(int i = 0; i<5;i++)
      {
            System.out.println(ran.nextInt());
      }
   }
}
```

Output :

64256704
1265771787
-1962940029
1372052

Date class :

The Date classes encapsulate information about a specific date and time.

It provides the following constructors.

Date()

Date(**long msec**)

Here, the first form returns an object that represent the current date and time.

The second form returns an object that represents the date and time msec in milliseconds after

Method	Description
Boolean after(Date d)	Returns true if d is after the current date. Otherwise, returns false.
Boolean before(Date d)	Returns true if d is before the current date. Otherwise, returns false.
Boolean equals(Date d)	Returns true if d has the same value as the current date. Otherwise, returns false.
Long getTime()	Returns the number of milliseconds since the epoch.
Void setTime (long msec)	Sets the date and time of the current object to represent msec milliseconds since the epoch.
String toString()	Returns the string equivalent of the date.

```java
import java.util.*;
public class Date_Demo
{
   public static void main(String args[])
   {
      Date dt = new Date();
      System.out.println(dt);

      Date epoch = new Date(0);
      System.out.println(epoch);
   }
}
```

Output :

Fri May 25 00:04:06 IST 2012
Thu Jan 01 05:30:00 IST 1970

```java
import java.util.Date;
public class date2
{
    public static void main(String[] args)
    {

     Date d1 = new Date();

     try
     {
      Thread.sleep(1000);
     }
     catch(Exception e){}

     Date d2 = new Date();

     System.out.println("First Date : " + d1);
     System.out.println("Second Date : " + d2);
     System.out.println("Is second date after first ? : " + d2.after(d1));

    }
}
```

Output :

First Date : Fri May 25 00:06:46 IST 2012
Second Date : Fri May 25 00:06:47 IST 2012
Is second date after first ? : true

```
import java.util.*;
public class date3
{
  public static void main(String args[])
  {
     Date date = new Date();
     System.out.println("Date is : " + date);
     System.out.println("Milliseconds since January 1, 1970, 00:00:00 GMT : " + date.getTime());
     Date epoch = new Date(0);
     date.setTime(10000);
     System.out.println("Time after 10 second " + epoch);
     System.out.println("Time after 10 second " + date);
     String st = date.toString();
     System.out.println(st);
  }
}
```

Output :

```
Date is : Fri May 25 00:12:52 IST 2012
Milliseconds since January 1, 1970, 00:00:00 GMT :
1337884972842
Time after 10 second Thu Jan 01 05:30:00 IST 1970
Time after 10 second Thu Jan 01 05:30:10 IST 1970
Thu Jan 01 05:30:10 IST 1970
```

The Calendar class allows you to interpret date and time information.

This class defines several integer constants that are used when you get or set components of the calendar. These are listed here.

AM	AM_PM	APRIL
AUGUST	DATE	DAY_OF_MONTH
DAY_OF_WEEK	DAY_OF_WEEK_IN_MONTH	DAY_OF_YEAR
DECEMBER	DST_OFFSET	ERA
FEBRUARY	FIELD_COUNT	FRIDAY
HOUR	HOUR_OF_DAY	JANUARY
JULY	JUNE	MARCH
MAY	MILLISECOND	MINUTE

MONDAY	MONTH	NOVEMBER
OCTOBER	PM	SATURADAY
SECOND	SEPTEMBER	SUNDAY
THURSDAY	TUESDAY	UNDERIMBER
WEDNESDAY	WEEK_OF_MONTH	WEEK_OF_YEAR
YEAR	ZONE_OFFSET	The Calendar class does not have public constructors. Instead, you may use the static getInstance() method to obtain a calendar initialized to the current date and time.

One of its forms is shown here:

Calendar getInstance()

```
import java.util.Calendar;
 public class Cal1
{
  public static void main(String[] args)
  {
     Calendar cal = Calendar.getInstance();

   System.out.println("DATE is : " + cal.get(cal.DATE));
   System.out.println("YEAR is : " + cal.get(cal.YEAR));
   System.out.println("MONTH is : " + cal.get(cal.MONTH));
   System.out.println("DAY OF WEEK is : " + cal.get(cal.DAY_OF_WEEK));
   System.out.println("WEEK OF MONTH is : " + cal.get(cal.WEEK_OF_MONTH));
   System.out.println("DAY OF YEAR is : " + cal.get(cal.DAY_OF_YEAR));
   System.out.println("DAY OF MONTH is : " + cal.get(cal.DAY_OF_MONTH));
   System.out.println("WEEK OF YEAR is : " + cal.get(cal.WEEK_OF_YEAR));
   System.out.println("HOUR is : " + cal.get(cal.HOUR));
   System.out.println("MINUTE is : " + cal.get(cal.MINUTE));
   System.out.println("SECOND is : " + cal.get(cal.SECOND));
   System.out.println("DAY OF WEEK IN MONTH is : " +
cal.get(cal.DAY_OF_WEEK_IN_MONTH));
   System.out.println("Era is : " + cal.get(cal.ERA));
   System.out.println("HOUR OF DAY is : " + cal.get(cal.HOUR_OF_DAY));
   System.out.println("MILLISECOND : " + cal.get(cal.MILLISECOND));
   System.out.println("AM_PM : " + cal.get(cal.AM_PM));// Returns 0 if AM and 1 if PM
  }
}
```

Output :

```
Date is : Fri May 25 00:21:14 IST 2012
Milliseconds since January 1, 1970, 00:00:00 GMT :
1337885474477
Time after 10 second Thu Jan 01 05:30:00 IST 1970
Time after 10 second Thu Jan 01 05:30:10 IST 1970
Thu Jan 01 05:30:10 IST 1970
```

The GregorianCalendar class is a subclass of Calendar.

It provides the logic to **manage date and time information** according to the rules of the Gregorian calendar.

This class provides following constructors:

GregorianCalendar()
GregorianCalendar(**int year, int month, int date**)
GregorianCalendar(**int year, int month, int date, int hour, int minute, int sec**)

GregorianCalendar(**int year, int month, int date, int hour, int minute**)

The first form creates an object initialized with the current date and time.

The other forms allow you to specify how various date and time components are initialized.

The class provides all of the method defined by Calendar and also adds the isLeapYear() method shown here:

Boolean isLeapYear() This method returns true if the current year is a leap year. Otherwise, it returns false.

```java
import java.util.*;
 public class gcal1
{
  public static void main(String[] args)
  {
   GregorianCalendar c1 = new GregorianCalendar() ;

   System.out.println("DATE is : " + c1.get(c1.DATE));
   System.out.println("YEAR is : " + c1.get(c1.YEAR));
   System.out.println("MONTH is : " + c1.get(c1.MONTH));
   System.out.println("DAY OF WEEK is : " + c1.get(c1.DAY_OF_WEEK));
   System.out.println("WEEK OF MONTH is : " + c1.get(c1.WEEK_OF_MONTH));
   System.out.println("DAY OF YEAR is : " + c1.get(c1.DAY_OF_YEAR));
   System.out.println("DAY OF MONTH is : " + c1.get(c1.DAY_OF_MONTH));
   System.out.println("WEEK OF YEAR is : " + c1.get(c1.WEEK_OF_YEAR));
   System.out.println("HOUR is : " + c1.get(c1.HOUR));
   System.out.println("MINUTE is : " + c1.get(c1.MINUTE));
   System.out.println("SECOND is : " + c1.get(c1.SECOND));
   System.out.println("DAY OF WEEK IN MONTH is : " +
c1.get(c1.DAY_OF_WEEK_IN_MONTH));
   System.out.println("Era is : " + c1.get(c1.ERA));
   System.out.println("HOUR OF DAY is : " + c1.get(c1.HOUR_OF_DAY));
   System.out.println("MILLISECOND : " + c1.get(c1.MILLISECOND));
   System.out.println("AM_PM : " + c1.get(c1.AM_PM));// Returns 0 if AM and 1 if PM  */
  }
}
```

Output :

Era is : 1
HOUR OF DAY is : 0
MILLISECOND : 780
AM_PM : 0

Math Class :

For scientific and engineering calculations, a variety of mathematical functions are required.

Java provides these functions in the Math class available in java.lang package.

The methods defined in Math class are given following:

Method	Description
Double sin(double x)	Returns the sine value of angle x in radians.
Double cos(double x)	Returns the cosine value of the angle x in radians
Double tan(double x)	Returns the tangent value of the angle x in radians
Double asin(double x)	Returns angle value in radians for arcsin of x
Double acos(double x)	Returns angle value in radians for arcos of x
Double atan(double x)	Returns angle value in radians for arctangent of x

Double exp(double x)	Returns exponential value of x
Double log(double x)	Returns the natural logarithm of x
Double pow(double x, double y)	Returns x to the power of y
Double sqrt(double x)	Returns the square root of x
Int abs(double n)	Returns absolute value of n
Double ceil(double x)	Returns the smallest wholoe number greater than or equal to x
Double floor(double x)	Returns the largest whole number less than or equal to x

Int max(int n, int m)	Returns the maximum of n and m
Int min(int n, int m)	Returns the minimum of n and m
Double rint(double x)	Returns the rounded whole number of x
Int round(float x)	Returns the rounded int value of x
Long round(double x)	Returns the rounded int value of x
Double random()	Returns a random value between 0 and 1.0
Double toRandians(double angle)	Converts the angle in degrees to radians
Double toDegrees(double angle)	Converts the angle in radians to degrees

```
public class Angles
{
   public static void main(String args[])
   {
      double theta = 120.0;
      System.out.println(theta + " degrees is " + Math.toRadians(theta) + " radians.");
      theta = 1.312;
      System.out.println(theta + " radians is " + Math.toDegrees(theta) + " degrees.");
   }
}
```

Output :

120.0 degrees is 2.0943951023931953 radians.
1.312 radians is 75.17206272116401 degrees.

Hashtable is a part of the **java.util** library and is a concrete implementation of a dictionary.

(Dictionary is a class that represents a key/value storage repository. Given a key and value, you can store the value in a Dictionary object. Once the value is stored, you can retrieve it by using its key.)

Hashtable stores key/value pairs in a hash table. When using a Hashtable, you specify an object that is used as a key, and the value that you want to link to that key.

The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.

The Hashtable constructors are shown here:

Hashtable()

Hashtable(**int size**)

The first constructor is the default constructor.

The second constructor creates a hash table that has an initial size specified by size.

The methods available with Hashtable are

Method	Description
Void clear()	Resets and empties the hash table.
Boolean containsKey(Object key)	Returns true if some key equals to key exists within the hash table. Returns false if the key isn't found.
Boolean containsValue(Object value)	Returns true if some value equal to value exists within the hash table. Returns false if the value isn't found.
Enumeration elements( )	Returns an enumeration of the values contained in the hash table.
Object get(Object key)	Returns the object that contains the value associated with key. If key is not in the hash table, a null object is returned.
Boolean isEmpty( )	Returns true if the hash table is empty; Returns false if it contains at least one key.

Enumeration keys( )	Returns an enumeration of the keys contained in the hash table.
Object put(Object key Object value)	Inserts a key and a value into the hash table.
Object remove(Object key)	Removes key and its value. Returns the value associated with key. If key is not in the hash table, a null object is returned.
Int size( )	Returns the number of entries in the hash table.
String toString( )	Returns the string equivalent of a hash table.
Enumeration keys( )	Returns an enumeration of the keys contained in the hash table.
Object put(Object key Object value)	Inserts a key and a value into the hash table.

```
import java.util.*;
 public class hash1
{
   public static void main(String args[])
   {
      Hashtable marks = new Hashtable();
      Enumeration names;
      Enumeration emarks;
      String str;
      int nm;

      // Checks wheather the hashtable is empty
      System.out.println("Is Hashtable empty " + marks.isEmpty());
      marks.put("Ram", 58);
      marks.put("Laxman", 88);
      marks.put("Bharat", 69);
      marks.put("Krishna", 99);
      marks.put("Janki", 54);

      System.out.println("Is Hashtable empty " + marks.isEmpty());
      // Creates enumeration of keys
      names = marks.keys();
      while(names.hasMoreElements())
      {
         str = (String) names.nextElement();
         System.out.println(str + ": " + marks.get(str));
      }
```

```
/*
    nm = (Integer) marks.get("Janki");
    marks.put("Janki", nm+15);
    System.out.println("Janki's new marks: " +  marks.get("Janki"));

    // Creates enumeration of values
    emarks = marks.elements();
    while(emarks.hasMoreElements())
    {
       nm = (Integer) emarks.nextElement();
       System.out.println(nm);
    }

    // Number of entries in a hashtable
    System.out.println("The number of entries in a table are " + marks.size());

    // Checking wheather the element available

    System.out.println("The element is their " + marks.containsValue(88));
    */
    // Removing an element from hashtable
```

```
System.out.println("========");
    marks.remove("Bharat");
    names = marks.keys();
    while(names.hasMoreElements())
    {
      str = (String) names.nextElement();
      System.out.println(str + ": " + marks.get(str));
    }
    // Returning an String equivalent of the Hashtable

    System.out.println("String " + marks.toString());
    // Emptying hashtable

    marks.clear();
    System.out.println("Is Hashtable empty " + marks.isEmpty());
  }
}
```

Output :

Laxman: 88

Janki: 54

Ram: 58

String {Krishna=99, Laxman=88, Janki=54, Ram=58}

Is Hashtable empty true

Everyone's journey is different. Don't compare your path to anyone else's.