Chapter - 12

The java.util Package

The java.util Package

This chapter covers the classes present in the java.util package. This package contains large number of classes and interfaces that supports major functionality. This package also contains predefined data structure classes called Collections in Java.

The following table contains the classes in **java.util** package

AbstractCollection	AbstractList	AbstractMap
AbstractSequentialList	AbstractSet	ArrayDeque
Arrays	BitSet	Calendar
Currency	Date	Dictionary
EnumSet	EventListenerProxy	EventObject
GregorianCalendar	HashMap	Hashtable
LinkedHashSet	LinkedList	ListResourceBoundle
observable	PriorityQueue	Properties
PropertyResourceBundle	Random	ResourceBundle
ServiceLoader	SimpleTimeZone	Stack
Timer	TimerTask	TimeZone
TreeSet	UUID	Vector
AbstractQueue	ArrayList	Collections
FormattableFlags	IdentityHashMap	Locale
Scanner	StringTokenizer	TreeMap
EnumMap	PropertyPermission	WeakHashMap

The following table contains interfaces in **java.util** package

Collection	Comparator	Deque	Enumeration
EventListener	Formattable	Iterator	List
ListIterator	Map	Map.Entry	NavigableMap
NavigableSet	Observer	Queue	RandomAccess
Set	SortedMap	SortedSet	

The StringTokenizer class

The **StringTokenizer** class is present in **java.util** package. This class can be used to break text into small parts called tokens. This process of dividing the text into parts is known as parsing. Parsing is the division of text into a set of discrete parts, or **tokens**, which in a certain sequence can convey a semantic meaning. The division of text into parts is based on ***delimiters,*** delimiters are characters based on the text is divided into parts i.e. delimiters are characters that separates tokens. Each character in the delimiters string is considered a valid delimiter—for example, **",;:"** sets the delimiters to a comma, semicolon, and colon. The default set of delimiters consists of the whitespace characters: space, tab, newline, and carriage return.

The set of delimiters (the characters that separate tokens) may be specified either at creation time or on a per-token basis.

The following table contains the constructors of **StringTokenizer** class

Constructors	Meaning
StringTokenizer(String str)	Constructs a string tokenizer for the specified string.(default delimiters are considered)
StringTokenizer(String str,String delim)	Constructs a string tokenizer for the specified string using given *delim.*
StringTokenizer(String str,String delim, boolean returnDelims)	Constructs a string tokenizer for the specified string using given *delim.* delimiters are returned along with tokens if *returnDelims* is *true* otherwise not.

In all versions of constructors, ***str*** is the string that will be tokenized. In the first version, the default delimiters are used. In the second and third versions, ***delim*** is a string that specifies the delimiters. In the third version, if ***returnDelims*** is **true**, then the delimiters are also returned as tokens when the string is parsed. Otherwise, the delimiters are not returned.

The following table contains the methods of **StringTokenizer** class

Methods	Meaning
int **countTokens**()	Calculates the number of times that this tokenizer's nextToken method can be called before it generates an exception.
Boolean **hasMoreElements**()	Tests if there are more tokens available from this tokenizer's string.
Boolean **hasMoreTokens**()	Tests if there are more tokens available from this tokenizer's string.
Object **nextElement**()	Returns the same value as the nextToken method, except that its declared return value is Object rather than String.
String **nextToken**()	Returns the next token from this string tokenizer.
String **nextToken**(String delim)	Returns the next token in this string tokenizer's string.

A program to demonstrate StringTokenizer

>edit token1.java

```
import java.util.StringTokenizer;
class token1
{
    public static void main(String argv[])
    {
        String str="apex=computer,wgl:AP;India";
        StringTokenizer st=new StringTokenizer(str, ";:,=");
        System.out.print("\n Total Tokens = "+
                                    st.countTokens() );
        System.out.print("\n Tokens are :");
        while(st.hasMoreTokens())
        {
            System.out.print("\n " + st.nextToken() );
        }
    }
}
```

Bin>javac token1.java

Bin>java token1

```
Total Tokens = 5
Tokens are :
apex
computer
wgl
AP
India
```

Explanation:

In the main(), **str** is stores with the string **"apex=computer,wgl:AP;India"**. The statement, **StringTokenizer st=new StringTokenizer(str, ";:,=");** creates an object of **StringTokenizer** class and **str** string divides into tokens(Apex computer wgl AP India) based on the delimiters(;:,=) and object is assigned to **st** reference variable. the **st** object contains an iterator (pointer to token) points to first token using which tokens can be accessed individually.

The **st.countTokens()** returns 5 because **st** contains **5** tokens.

In the while loop, **st.hasMoreTokens()** return true if tokens are available for visiting and in the loop, **st.nextToken()** returns the current token and moves the iterator to next token. The while loop prints all the tokens one after the other as shown in the above output.

The Date Class

The **Date** class encapsulates the current date and time. The date functions of **Date** class present in the original version of Java 1.0 was moved to the **Calendar** and **DateFormat** classes from Java 1.1. Some additional functions also added into **Date** class from Java 1.1.

The following table describes the constructors of **Date** class

Constructors	Meaning
Date()	Allocates a Date object and initializes it so that it represents the time at which it was allocated, measured to the nearest millisecond.
Date(long date)	Allocates a Date object and initializes it to represent the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.

The following table describes methods of Date class

Method	Meaning
boolean after(Date date)	Returns true if the invoking Date object contains a date that is later than the one specified by date. Otherwise, it returns false.
boolean before(Date date)	Returns true if the invoking Date object contains a date that is earlier than the one specified by date. Otherwise, it returns false.
Object clone()	Duplicates the invoking Date object.
int compareTo(Date date)	Compares the value of the invoking object with that of date. Returns 0 if the values are equal. Returns a negative value if the invoking object is earlier than date. Returns a positive value if the invoking object is later than date. (Added by Java 2)
int compareTo(Object obj)	Operates identically to compareTo(Date)if obj is of class Date. Otherwise, it throws a ClassCastException. (Added by Java 2)
boolean equals(Object date)	Returns true if the invoking Date object contains the same time and date as the one specified by date. Otherwise, it returns false.

long getTime()	Returns the number of milliseconds that have elapsed since January 1 1970.
int hashCode()	Returns a hash code for the invoking object.
void setTime(long time)	Sets the time and date as specified by time, which represents an elapsed time in milliseconds from midnight, January 1, 1970.
String toString()	Converts the invoking Date object into a string and returns the result.

Bin>edit date1.java

```
class date1
{
      public static void main(String argv[])
      {
            Date d=new Date();
            System.out.print("\n date =" +d );
            long timemsec=d.getTime();
            System.out.print("\n Milliseconds since
                  Jan 1, 1970 GMT = "+timemsec);

      }
}
```

Bin>javac date1.java
Bin>java date1

-Execute and get the output

The Random class

Random numbers in java can be generated either by using the ***Random*** class in **java.util** package or by using the ***random()*** method in the **Math** class in **java.lang** package.

In both these approaches, we only get a pseudo random number and not a true random number because it is all generated based on some mathematical computation, and hence the number can be predicted if the logic behind the computation is known. In spite of this, we can use any of these two approaches itself for random number generation for most of our application needs.

Using the Random class

The Random class in **java.util** package can be used in order to generate random numbers. A seed value can be specified, which would be used for generating random numbers. This seed value can be specified while creating the Random object. If it is not specified, then the default seed value would be the current time in a day.

There are several methods such as **next(), nextInt(), nextDouble(), nextFloat(), nextLong(), nextBytes(), nextBoolean()** and **nextGaussian()** etc.. which can be used to generate random numbers of different data types.

The following table contains the constructors of **Random** class

Constructors	Meaning
Random()	Creates a new random number generator.
Random(long seed)	Creates a new random number generator using a single long seed:

The following table contains the methods of **Random** class

Methods	Meaning
protected int **next**(int bits)	Generates the next pseudorandom number.
boolean **nextBoolean**()	Returns the next pseudorandom, uniformly distributed boolean value from this random number generator's sequence.
void **nextBytes**(byte[] bytes)	Generates random bytes and places them into a user-supplied byte array.
double **nextDouble**()	Returns the next pseudorandom, uniformly distributed double value between 0.0 and 1.0 from this random number generator's sequence.
float **nextFloat**()	Returns the next pseudorandom, uniformly distributed float value between 0.0 and 1.0 from this random number generator's sequence.
double **nextGaussian**()	Returns the next pseudorandom, Gaussian ("normally") distributed double value with mean 0.0 and standard deviation 1.0 from this random number generator's sequence.

int **nextInt**()	Returns the next pseudorandom, uniformly distributed int value from this random number generator's sequence.
int **nextInt**(int n)	Returns a pseudorandom, uniformly distributed int value between 0 (inclusive) and the specified value (exclusive), drawn from this random number generator's sequence.
long **nextLong**()	Returns the next pseudorandom, uniformly distributed long value from this random number generator's sequence.
void **setSeed**(long seed)	Sets the seed of this random number generator using a single long seed.

A simple example program for generating random numbers using the Random class,

Bin>edit rand1.java

```
import java.util.Random;
class rand1
{
      public static void main(String argv[])
      {
            Random r=new Random();
            int i;
            for(i=1;i<=5;i++)
            {
                  System.out.print("\t" + r.nextInt(100));
            }
            byte b[]=new byte[10];
            r.nextBytes(b);
            System.out.print("\n values of b array\n");
            for(i=0;i<10;i++)
            {
                  System.out.print("\t"+b[i]);
            }
      }
}
```

Bin>javac rand1.java

Bin>java rand1

```
values of b array
66  93  34  40  63
98  -128  6  108  96  119  -36  -40  105
-116
```

The Calendar class

Calendar is an abstract base class for converting between a **Date** object and a set of integer fields such as **YEAR**, **MONTH**, **DAY**, **HOUR**, and so on.

Subclasses of **Calendar** interpret a **Date** according to the rules of a specific calendar system. The platform provides one concrete subclass of **Calendar**: **GregorianCalendar**. Future subclasses could represent the various types of lunar calendars in use in many parts of the world.

Like other locale-sensitive classes, Calendar provides a class method, getInstance(), for getting a generally useful object of this type. Calendar's getInstance() method returns a **Calendar** object whose time fields have been initialized with the current date and time:

Calendar rightNow = Calendar.getInstance();

A **Calendar** object can produce all the time field values needed to implement the date-time formatting for a particular language and calendar style (for example, Japanese-Gregorian, Japanese-Traditional). Calendar defines the range of values returned by certain fields, as well as their meaning. For example, the first month of the year has value **MONTH == JANUARY** for all calendars. Other values are defined by the concrete subclass, such as ERA and YEAR. See individual field documentation and subclass documentation for details.

Following table shows variables of **Calendar** class.

Fields	Meaning
Static int **AM**	Value of the AM_PM field indicating the period of the day from midnight to just before noon.
Static int **AM_PM**	Field number for get and set indicating whether the HOUR is before or after noon.
Static int **APRIL**	Value of the MONTH field indicating the fourth month of the year.
protected Boolean **areFieldsSet**	True if fields[] are in sync with the currently set time.

Static int **AUGUST**	Value of the MONTH field indicating the eighth month of the year.
Static int **DATE**	Field number for get and set indicating the day of the month.
Static int **DAY_OF_MONTH**	Field number for get and set indicating the day of the month.
Static int **DAY_OF_WEEK**	Field number for get and set indicating the day of the week.
Static int **DAY_OF_WEEK_IN_MONTH**	Field number for get and set indicating the ordinal number of the day of the week within the current month.
Static int **DAY_OF_YEAR**	Field number for get and set indicating the day number within the current year.
Static int **DECEMBER**	Value of the MONTH field indicating the twelfth month of the year.
Static int **DST_OFFSET**	Field number for get and set indicating the daylight savings offset in milliseconds.
Static int **ERA**	Field number for get and set indicating the era, e.g., AD or BC in the Julian calendar.
Static int **FEBRUARY**	Value of the MONTH field indicating the second month of the year.
Static int **FIELD_COUNT**	The number of distinct fields recognized by get and set.
protected int[]**fields**	The field values for the currently set time for this calendar.
Static int **FRIDAY**	Value of the DAY_OF_WEEK field indicating Friday.
static int **HOUR**	Field number for get and set indicating the hour of the morning or afternoon.
static int **HOUR_OF_DAY**	Field number for get and set indicating the hour of the day.
protected boolean[]**isSet**	The flags which tell if a specified time field for the calendar is set.
protected boolean **isTimeSet**	True if then the value of time is valid.
static int **JANUARY**	Value of the MONTH field indicating the first month of the year.
static int **JULY**	Value of the MONTH field indicating the seventh month of the year.
static int **JUNE**	Value of the MONTH field indicating the sixth month of the year.

static int **MARCH**	Value of the MONTH field indicating the third month of the year.
static int **MAY**	Value of the MONTH field indicating the fifth month of the year.
static int **MILLISECOND**	Field number for get and set indicating the millisecond within the second.
static int **MINUTE**	Field number for get and set indicating the minute within the hour.
static int **MONDAY**	Value of the DAY_OF_WEEK field indicating Monday.
static int **MONTH**	Field number for get and set indicating the month.
static int **NOVEMBER**	Value of the MONTH field indicating the eleventh month of the year.
static int **OCTOBER**	Value of the MONTH field indicating the tenth month of the year.
static int **PM**	Value of the AM_PM field indicating the period of the day from noon to just before midnight.
static int **SATURDAY**	Value of the DAY_OF_WEEK field indicating Saturday.
static int **SECOND**	Field number for get and set indicating the second within the minute.
static int **SEPTEMBER**	Value of the MONTH field indicating the ninth month of the year.
static int **SUNDAY**	Value of the DAY_OF_WEEK field indicating Sunday.
static int **THURSDAY**	Value of the DAY_OF_WEEK field indicating Thursday.
protected long **time**	The currently set time for this calendar, expressed in milliseconds after January 1, 1970, 0:00:00 GMT.
static int **TUESDAY**	Value of the DAY_OF_WEEK field indicating Tuesday.
static int **UNDECIMBER**	Value of the MONTH field indicating the thirteenth month of the year.
static int **WEDNESDAY**	Value of the DAY_OF_WEEK field indicating Wednesday.
static int **WEEK_OF_MONTH**	Field number for get and set indicating the week number within the current month.
static int **WEEK_OF_YEAR**	Field number for get and set indicating the week number within the current year.

static int **YEAR**	Field number for get and set indicating the year.
static int **ZONE_OFFSET**	Field number for get and set indicating the raw offset from GMT in milliseconds.

Following table shows constructors of **Calendar** class.

Constructors	**Meaning**
protected **Calendar**()	Constructs a Calendar with the default time zone and locale.
protected **Calendar**(TimeZone zone, Locale aLocale)	Constructs a calendar with the specified time zone and locale.

Following table shows methods of **Calendar** class.

Methods	**Meaning**
boolean **equals**(Object obj)	Compares this calendar to the specified object.
int **get**(int field)	Gets the value for a given time field.
int **getActualMaximum**(int field)	Return the maximum value that this field could have, given the current date.
int **getActualMinimum**(int field)	Return the minimum value that this field could have, given the current date.
int **getFirstDayOfWeek**()	Gets what the first day of the week is; e.g., Sunday in US, Monday in France.
abstract int **getGreatestMinimum** (int field)	Gets the highest minimum value for the given field if varies.
static Calendar **getInstance**()	Gets a calendar using the default time zone and locale.
abstract int **getLeastMaximum**(int field)	Gets the lowest maximum value for the given field if varies.
abstract int **getMaximum**(int field)	Gets the maximum value for the given time field.
int **getMinimalDaysInFirst Week**()	Gets what the minimal days required in the first week of the year are; e.g., if the first week is defined as one that contains the first day of the first month of a year, getMinimalDaysInFirstWeek returns 1.

abstract int **getMinimum**(int field)	Gets the minimum value for the given time field.
Date **getTime**()	Gets this Calendar's current time.
long **getTimeInMillis**()	Gets this Calendar's current time as a long.
TimeZone **getTimeZone**()	Gets the time zone.
void **set**(int field, int value)	Sets the time field with the given value.
void **set**(int year, int month, int date)	Sets the values for the fields year, month, and date.
void **set**(int year, int month, int date, int hour, int minute)	Sets the values for the fields year, month, date, hour, and minute.
void **set**(int year, int month, int date, int hour, int minute, int second)	Sets the values for the fields year, month, date, hour, minute, and second.
void **setFirstDayOfWeek**(int value)	Sets what the first day of the week is; e.g., Sunday in US, Monday in France.
void **setMinimalDaysInFirstWeek**(int value)	Sets what the minimal days required in the first week of the year are; For example, if the first week is defined as one that contains the first day of the first month of a year, call the method with value 1.
void **setTime**(Date date)	Sets this Calendar's current time with the given Date.
void **setTimeInMillis**(long millis)	Sets this Calendar's current time from the given long value.
void **setTimeZone**(TimeZone value)	Sets the time zone with the given time zone value.
String **toString**()	Return a string representation of this calendar.

Bin>edit calendar1.java

```
import java.util.Date;
import java.util.Calendar;
import java.text.SimpleDateFormat;
import java.util.*;
class calendar1
{
  private static void CalendarTimemethod()
  {
    Date date = Calendar.getInstance().getTime();
```

```
        System.out.println("Current date and time is: " + date);
        System.out.println();
    }
    private static void SimpleDateFormatmethod()
    {
        Calendar date = Calendar.getInstance();
        SimpleDateFormat dateformatter =
        new SimpleDateFormat("E yyyy.MM.dd 'at' hh:mm:ss a zzz");
        System.out.println("Current date and time in simple
        date format:"+dateformatter.format(date.getTime()));
        System.out.println();
    }
    private static void Getcalendermethods()
    {
        System.out.println("Various get methods of the
        calendar class:");
        Calendar calender = Calendar.getInstance();
        System.out.println("Year    : " +
              calender.get(Calendar.YEAR));
        System.out.println("Month     : " +
              calender.get(Calendar.MONTH));
        System.out.println("Day of Month  : " +
              calender.get(Calendar.DAY_OF_MONTH));
        System.out.println("Day of Week   : " +
              calender.get(Calendar.DAY_OF_WEEK));
        System.out.println("Day of Year   : " +
              calender.get(Calendar.DAY_OF_YEAR));
        System.out.println("Week of Year  : " +
              calender.get(Calendar.WEEK_OF_YEAR));
        System.out.println("Week of Month  : " +
              calender.get(Calendar.WEEK_OF_MONTH));
        System.out.println("Day of the Week in Month : " +
              calender.get(Calendar.DAY_OF_WEEK_IN_MONTH));
        System.out.println("Hour : " +
              calender.get(Calendar.HOUR));
        System.out.println("AM PM : " +
              calender.get(Calendar.AM_PM));
        System.out.println("Hour of the Day : " +
              calender.get(Calendar.HOUR_OF_DAY));
        System.out.println("Minute : " +
              calender.get(Calendar.MINUTE));
```

```
        System.out.println("Second : " +
                    calender.get(Calendar.SECOND));
        System.out.println();
    }

    public static void main(String[] args)
    {
        CalendarTimemethod();
        SimpleDateFormatmethod();
        Getcalendermethods();
    }
}
```

Bin>javac calendar1.java
Bin>java calendar1

```
Current date and time is: Mon Jan 03 10:59:03 GMT 2011

Current date and time in simple date format: Mon 2011.01.03 at 10:59:03 AM GMT

Various get methods of the calendar class:
Year   : 2011
Month    : 0
Day of Month  : 3
Day of Week  : 2
Day of Year  : 3
Week of Year  : 2
Week of Month  : 2
Day of the Week in Month : 1
Hour : 10
AM PM : 0
Hour of the Day : 10
Minute : 59
Second : 3
```