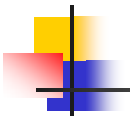


Programming Java

The Java Class Libraries

Incheon Paik

Contents

The Random Class

Random Constructor

Random()
Random(long seed)
seed: a value to initialize the random number generator

Methods Defined by Random

void nextBytes(byte buffer[])
double nextDouble()
float nextFloat()
double nextGaussian()
int nextInt()
long nextLong()
void setSeed(long seed)

http://java.sun.com/j2se/1.4.2/docs/api/java/util/Random.html

```
import java.util.*;
class RandomInts {
  public static void main(String args[]) {
    // Create Random Number Generator
    Random generator = new Random();
    // Generate and display 10 random integers
    for (int i = 0; i < 10; i++)
      System.out.println(generator.nextInt());
  }
}
```

```
Result :
-237943534
2142887930
-2034707047
1221578671
-310733567
1144698518
-126297085
-2056908663
168018234
510116388
```


The Date Class

Date Constructor

Date()
Date(long msec)
msec: miliseconds after the epoch.

Methods Defined by Date

boolean after(Date d)
boolean before(Date d)
boolean equals(Object d)
long getTime()
void setTime(long msec)
String toString()

```
import java.util.*;

class DateDemo {

  public static void main(String args[]) {

    Date currentDate = new Date();

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

```
Result :
Fri May 07 11:43:30 GMT+09:00 2004
Thu Jan 01 09:00:00 GMT+09:00 1970
```

Refer to
http://java.sun.com/j2se/1.4.2/docs/api/java/util/Date.html

The Calendar Class and GregorianCalendar Class

To get Calendar Object

Calendar now = Calendar.getInstance();

GregorianCalendar Constructor

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)

isLeapYear() Method

boolean isLeapYear(int year)

Methods in Calendar Class

abstract boolean after(Object calendarObj)

abstract boolean before(Object calendarObj)

abstract boolean equals(Object calendarObj)

final int get(int calendarField)

static Calendar getInstance()

final Date getTime()

final void set(int year, int month, int date, int hour, int minute, int second)

final void setTime(Date d)

Refer to the
http://java.sun.com/j2se/1.4.2/docs/api/java/util/Calendar.html

The Calendar Class and GregorianCalendar Class

```
import java.util.*;

class CalendarDemo {

  public static void main(String args[]) {

    Calendar calendar = Calendar.getInstance();
    System.out.println(calendar.get(Calendar.YEAR));
    System.out.println(calendar.get(Calendar.HOUR));
    System.out.println(calendar.get(
      Calendar.HOUR_OF_DAY));
    System.out.println(calendar.get(Calendar.MINUT
E));
  }
}
```

Result :

2004
10
22
58

Calendar Instance

The Vector Class and Enumeration Class

Vector Constructor

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

hasMoreElements(), nextElement() Method

boolean hasMoreElements()
Object nextElement()

Refer to the
http://java.sun.com/j2se/1.4.2/docs/api/java/util/Vector.html

Methods in Vector Class

void addElement(Object obj)
int capacity()
Object clone()
boolean contains(Object obj)
void copyInto(Object array[])
Object elementAt(int index)
Enumeration elements()
void ensureCapacity(int minimum)
Object firstElement()
int indexOf(Object obj)
int indexOf(Object obj, int index)
void insertElementAt(Object obj, int index)
boolean isEmpty(); Object lastElement()
int lastIndexOf(Object obj)
int lastIndexOf(Object obj, int index)
void removeAllElements()
boolean removeElement(Object obj)
void removeElementAt(int index)
void setElementAt(Object obj, int index)
void setSize(int size); int size()
String toString(); void trimToSize()

Vector Demo

```
class VectorDemo {

  public static void main(String args[]) {

    // Create a vector and its elements
    Vector vector = new Vector();
    vector.addElement(new Integer(5));
    vector.addElement(new Float(-14.14f));
    vector.addElement(new String("Hello"));
    vector.addElement(new Long(120000000));
    vector.addElement(new Double(-23.45e-11));

    // Display the  vector elements
    System.out.println(vector);

    // Insert an element into the vector
    String s = new String("String to be inserted");
    vector.insertElementAt(s, 1);
    System.out.println(vector);

    // Remove an element from the vector
    vector.removeElementAt(3);
    System.out.println(vector);
  }
}
```

Result :
[5, -14.14, Hello, 120000000, -2.345E-10]
[5, String to be inserted, -14.14, Hello, 120000000, -2.345E-10]
[5, String to be inserted, -14.14, 120000000, -2.345E-10]

Body of run method

Vector

The Stack Class

Stack

LIFO(Last-In-First-Out) Type Data Structure

Methods in Vector Class

boolean empty()
Object peek() throws EmptyStackException
Object pop() throws EmptyStackException
Object push(Object obj)
int search(Object obj)

Refer to the
http://java.sun.com/j2se/1.4.2/docs/api/java/util/Stack.html

Run java PushPop 1 2 3 4 5

```
import java.util.*;

class PushPop {

  public static void main(String args[]) {

    Stack stack = new Stack();


      for (int i = 0; i < args.length; i++)
       stack.push(new Integer(args[i]));


       while(!stack.empty()) {
       Object obj = stack.pop();
       System.out.println(obj);
     }
   }
}
```

Result
5
4
3
2
1

The Hashtable Class

Hashtable

Associative Array

Methods in Hashtable Class

boolean contains(Object v) throws NullPointerException
boolean containsAKey(Object k)
boolean containsValue(Object v)
Enumeration elements()
Object get(Object k)
boolean isEmpty()
Enumeration keys()
Object put(Object k, Object v) throws NullPointerException

Refer to the
http://java.sun.com/j2se/1.4.2/docs/api/java/util/Hashtable.html

Result #1
key = banana; value = yellow
key = apple; value = red
key = orange; value = orange

```
class HashtableDemo {

  public static void main(String args[]) {

    Hashtable hashtable = new Hashtable();
    hashtable.put("apple", "red");
    hashtable.put("strawberry", "red");
    hashtable.put("lime", "green");
    hashtable.put("banana", "yellow");
    hashtable.put("orange", "orange");

    Enumeration e = hashtable.keys();
    while(e.hasMoreElements()) {
      Object k = e.nextElement();
      Object v = hashtable.get(k);
      System.out.println("key = " + k + "; value = " + v);
    }
    System.out.print("¥nThe color of an apple is: ");
    Object v = hashtable.get("apple");
    System.out.println(v);
  }
}
```

Result #2
key = lime; value = green
key = strawberry; value = red

The color of an apple is: red

The StringTokenizer Class

StringTokenizer Constructor

StringTokenizer(String str)

StringTokenizer(String str, String delimiters)

StringTokenizer(String str, String delimiters, boolean delimitersAreTokens)

Methods in StringTokenizer Class

int countTokens()

boolean hasMoreTokens()

String nextToken()

String nextToken(String delimiters)

Refer to the
http://java.sun.com/j2se/1.4.2/docs/api/java/util/StringTokenizer.html

```
class StringTokenizerDemo {

  public static void main(String args[]) {

    String str =
      "123/45.6/-11.2/41/-90.1/100/99.99/-50/-20";
    StringTokenizer st = new StringTokenizer(str, "/");
    while(st.hasMoreTokens()) {
      String s = st.nextToken();
      System.out.println(s);
    }
  }
}
```

```
Result
123
45.6
-11.2
41
-90.1
100
99.99
-50
-20
```


The split method of the String Class

To parse a string, we can also use the split method of the String class. Recently, the split method is preferred to the StringTokenizer class, because it can use more general expression for parsing: regular expression, and provide more useful functions.

Split Method in theString Class

String[] split(String regex)

: Splits this string around matches of the given regular expression.

String[] split(String regex, int limit)

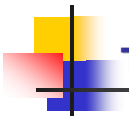
Refer to the
http://java.sun.com/j2se/1.5.0/docs/api/java/lang/String.html

```
public class StringTokenizing {
  public static void main(String[] args) {
    String text = "To be or not to be, that is the question.";
// String to be segmented
    String delimiters = "[, .]";
    // Analyze the string
      String[] tokens = text.split(delimiters);
      System.out.println("Number of tokens: " + tokens.len
gth);
      for(String token : tokens) {
        System.out.println(token);
      }
  }
}
```

```
Result
Number of tokens: 11
To
be
or
not
to
be

that
is
the
question
```

Because of two successive delimiters!



The Collections Framework

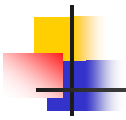
The Java collection framework is a set of generic types that are used to create collection classes that support various ways to store and manage objects of any kind in memory. A generic type for collection of objects: To get static checking by the compiler for whatever types of objects to want to manage.

Generic Types

Generic Class / Interface Type	Description
The **Iterator<T>** interface type	Declares methods for iterating through elements of a collection, one at a time.
The **Vector<T>** type	Supports an array-like structure for storing any type of object. The number of objects to be stored increases automatically as necessary.
The **Stack<T>** type	Supports the storage of any type of object in a pushdown stack.
The **LinkedList<T>** type	Supports the storage of any type of object in a doubly-linked list, which is a list that you can iterate though forwards or backwards.
The **HashMap<K,V>** type	Supports the storage of an object of type V in a hash table, sometimes called a map. The object is stored using an associated key object of type K. To retrieve an object you just supply its associated key.

Collections of Objects

- Three Main Types of Collections
  - Sets
  - Sequences
  - Maps
- Sets
  - The simple kinds of collection
  - The objects are not ordered in any particular way.
  - The objects are simply added to the set without any control over where they go.



Collections of Objects

- Sequences
  - The objects are stored in a linear fashion, not necessarily in any particul ar order, but in an arbitrary fixed sequence with a beginning and an end.
  - Collections generally have the capability to expand to accommodate as many elements as necessary.
  - The various types of sequence collections
    - Array or Vector
    - LinkedList
    - Stack
    - Queue

Collections of Objects

- Maps
  - Each entry in the collection involves a pair of objects.
  - A map is also referred to sometimes as a **dictionary**.
  - Each object that is stored in a map has an associated **key** object, and the object and its key are stored together as a “name-value” pair.

Iterators

Iterator<> interface

T next()
boolean hasNext()
void remove()

```
MyClass item;

while(iter.hasNext()) {
  item = iter.next();
  // Do something with item ….
}
```


List Iterators

ListIterator<> interface

T next()
boolean hasNext()
int nextIndex()
T previous()
boolean hasPrevious()
int previousIndex()

ListIterator object

void remove()
void add(T obj)
void set(T obj)

Collection Classes

Classes in Sets:

HashSet<T>
LinkedHashSet<T>
TreeSet<T>
EnumSet<T extends Enum<T>>

Classes in Lists:

Vector<T>
Stack<T>
LinkedList<T>
ArrayList<T>

Collection Classes

Class in Queues:

PriorityQueue<T>

Classes in Maps:

Hashtable<K,V>
HashMap<K,V>
LinkedHashMap<K,V>
WeakHashMap<K,V>
IdentityHashMap<K,V>
TreeMap<K,V>

- For the manner in which generic types representing sets, lists, and queues are related, you can refer to the Horton book, on page 613.

- For the parameterized types that define maps of various kinds, you can refer to the Horton book, on page 613.

Autoboxing and Auto-Unboxing of Primitive Types

- Converting between primitive types, like int, boolean, and their equivalent object-based c ounterparts like Integer and Boolean, can require unnecessary amounts of extra coding, e specially if the conversion is only needed for a method call to the Collections API, for ex- a mple.
The autoboxing and auto-unboxing of Java primitives produces code that is more concise and easier to follow. In the next example an int is being stored and then retrieved from an ArrayList. The 5.0 version leaves the conversion required to transition to an Integer and back in the compiler.

Before

```
ArrayList list = new ArrayList();
list.add(0, new Integer(42));
int total = ((Integer)list.get(0)).intValue();
```

After

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(0, 42);
int total = list.get(0);
```


Using a Vector

```
import java.util.Vector;

public class TrySimpleVector {

  public static void main(String args[]) {

   Vector<String> names = new Vector<String>();
   String[] firstnames = { “Jack”, “Jill”, “John”, “Joan”, “Jeremiah”, “Josephine”};

   // Add the names to the vector
   for(String firstname : firstnames) {
     names.add(firstname);
   }

   // List the contents of the vector
   for(String name : names) {
     System.out.println(name);
   }
  }
}
```

```
java.util.Iterator<String> iter = names.iterator();
while(iter.hasNext()) {
  System.out.println(iter.next());
}
```

The various samples are at

/home/course/prog3/java2-1.5/Ch14/

Exercise

- **Step 1 (Extract Contents from a File)**

Use the StringTokenizer or the split method of the String class
The Slides 11, 12, and refer to the example codes in exercise page

- **Step 2 (An Information Summarizer Using the Utility Classes)**

Use the StringTokenizer, the Vector, the Hashtable
The Slides 7, 8, 10, 11

To read the file, refer to the next

```
import java.io.*;
class BRDemo {
  public static void main(String args[]) {
    try {
      FileReader fr =
        new FileReader(args[0]);
      BufferedReader br =
        new BufferedReader(fr);
      String s;
      while((s = br.readLine()) != null) {
        System.out.println(s);
      }
    fr.close();
    }
    catch (Exception e) {
      System.out.println("Exception: " + e);
    }
  }
}
```


Exercise

- **Step 3 (Information Summarizer Using the Generic Class Types for Collections)**

Refer to Slides 13 – 19, and the example programs at
/home/course/prog3/java2-1.5/Ch14/TryVector
and
/home/course/prog3/java2-1.5/Ch14/TryHashMap