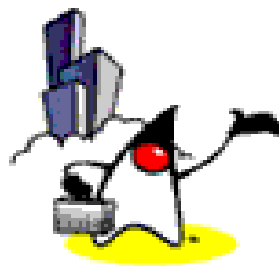




Tour of java.lang.* & java.util.* Classes



Topics

- *Math* Class
- *String* and the *StringBuffer* Class
- *Wrapper* Classes
- *Process* and the *Runtime* Class
- *System* Class



Math Class

The *Math* Class

- Provides predefined constants and methods for performing different mathematical operations
- Methods:

Math Methods
<code>public static double abs(double a)</code>
Returns the positive value of the parameter. An overloaded method. Can also take in a float or an integer or a long integer as a parameter, in which case the return type is either a float or an integer or a long integer, respectively.
<code>public static double random()</code>
Returns a random positive value greater than or equal to 0.0 but less than 1.0.
<code>public static double max(double a, double b)</code>
Returns the larger value between two <i>double</i> values, <i>a</i> and <i>b</i> . An overloaded method. Can also take in float or integer or long integer values as parameters, in which case the return type is either a float or an integer or a long integer, respectively.
<code>public static double min(double a, double b)</code>
Returns the smaller value between two <i>double</i> values, <i>a</i> and <i>b</i> . An overloaded method. Can also take in float or integer or long integer values as parameters, in which case the return type is either a float or an integer or a long integer, respectively.
<code>public static double ceil(double a)</code>
Returns the smallest integer that is greater than or equal to the specified parameter <i>a</i> .
<code>public static double floor(double a)</code>
Returns the largest integer that is lesser than or equal to the specified parameter <i>a</i> .



The *Math* Class: Methods

Math Methods
<code>public static double exp(double a)</code>
Returns Euler's number <i>e</i> raised to the power of the passed argument <i>a</i> .
<code>public static double log(double a)</code>
Returns the natural logarithm (base <i>e</i>) of <i>a</i> , the <i>double</i> value parameter.
<code>public static double pow(double a, double b)</code>
Returns the <i>double</i> value of <i>a</i> raised to the <i>double</i> value of <i>b</i> .
<code>public static long round(double a)</code>
Returns the nearest <i>long</i> to the given argument. An overloaded method. Can also take in a <i>float</i> as an argument and returns the nearest <i>int</i> in this case.
<code>public static double sqrt(double a)</code>
Returns the square root of the argument <i>a</i> .
<code>public static double sin(double a)</code>
Returns the trigonometric sine of the given angle <i>a</i> .
<code>public static double toDegrees(double angrad)</code>
Returns the degree value approximately equivalent to the given radian value.
<code>public static double toRadians(double angdeg)</code>
Returns the radian value approximately equivalent to the given degree value.



The *Math* Class: Example

```
1 class MathDemo {
2     public static void main(String args[]) {
3         System.out.println("absolute value of -5: " +
4                               Math.abs(-5));
5         System.out.println("absolute value of 5: " +
6                               Math.abs(-5));
7         System.out.println("random number(max is 10): " +
8                               Math.random()*10);
9         System.out.println("max of 3.5 and 1.2: " +
10                               Math.max(3.5,1.2));
11        System.out.println("min of 3.5 and 1.2: " +
12                               Math.min(3.5,1.2));
13    }
14 }
15 //continued...
```



The *Math* Class: Example

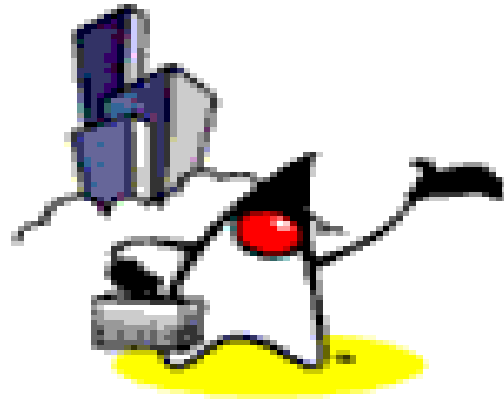
```
12      System.out.println("ceiling of 3.5: " +  
13                          Math.ceil(3.5)) ;  
14      System.out.println("floor of 3.5: " +  
15                          Math.floor(3.5)) ;  
16      System.out.println("e raised to 1: " +  
17                          Math.exp(1)) ;  
18      System.out.println("log 10: " + Math.log(10)) ;  
19      System.out.println("10 raised to 3: " +  
20                          Math.pow(10,3)) ;  
21  //continued...
```



The *Math* Class: Example

```
22      System.out.println("rounded off value of pi: " +
23                          Math.round(Math.PI));
24      System.out.println("square root of 5 = " +
25                          Math.sqrt(5));
26      System.out.println("10 radian = " +
27                          Math.toDegrees(10) + " degrees");
28      System.out.println("sin(90): " +
29                          Math.sin(Math.toRadians(90)));
30  }
31 }
```





String Class

The *String* Class

- Definition:
 - Represents combinations of character literals
 - Using Java, strings can be represented using:
 - Array of characters
 - The *String* class
 - Note: A *String* object is different from an array of characters!
- String constructors
 - 11 constructors



The *String* Class: Constructors

```
1 class StringConstructorsDemo {
2     public static void main(String args[]) {
3         String s1 = new String();           //empty string
4         char chars[] = { 'h', 'e', 'l', 'l', 'o' };
5         String s2 = new String(chars);      //s2="hello";
6         byte bytes[] = { 'w', 'o', 'r', 'l', 'd' };
7         String s3 = new String(bytes);      //s3="world"
8         String s4 = new String(chars, 1, 3);
9         String s5 = new String(s2);
10        String s6 = s2;
11    //continued
```



The *String* Class: Constructors

```
12      System.out.println(s1) ;  
13      System.out.println(s2) ;  
14      System.out.println(s3) ;  
15      System.out.println(s4) ;  
16      System.out.println(s5) ;  
17      System.out.println(s6) ;  
18  }  
19 }
```



The *String* Class: Methods

String Methods
<code>public char charAt(int index)</code>
Returns the character located in the specified <i>index</i> .
<code>public int compareTo(String anotherString)</code>
Compares this string with the specified parameter. Returns a negative value if this string comes lexicographically before the other string, 0 if both of the strings have the same value and a positive value if this string comes after the other string lexicographically.
<code>public int compareToIgnoreCase(String str)</code>
Like <code>compareTo</code> but ignores the case used in this string and the specified string.
<code>public boolean equals(Object anObject)</code>
Returns true if this string has the same sequence of characters as that of the <i>Object</i> specified, which should be a <i>String</i> object. Otherwise, if the specified parameter is not a <i>String</i> object or if it doesn't match the sequence of symbols in this string, the method will return false.
<code>public boolean equalsIgnoreCase(String anotherString)</code>
Like <code>equals</code> but ignores the case used in this string and the specified string.
<code>public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)</code>
Gets the characters from this string starting at the <i>srcBegin</i> index up to the <i>srcEnd</i> index and copies these characters to the <i>dst</i> array starting at the <i>dstBegin</i> index.



The *String* Class: Methods

String Methods
<code>public int length()</code>
Returns the length of this string.
<code>public String replace(char oldChar, char newChar)</code>
Returns the string wherein all occurrences of the <i>oldChar</i> in this string is replaced with <i>newChar</i> .
<code>public String substring(int beginIndex, int endIndex)</code>
Returns the substring of this string starting at the specified <i>beginIndex</i> index up to the <i>endIndex</i> index.
<code>public char[] toCharArray()</code>
Returns the character array equivalent of this string.
<code>public String trim()</code>
Returns a modified copy of this string wherein the leading and trailing white space are removed.
<code>public static String valueOf(-)</code>
Takes in a simple data type such as boolean, integer or character, or it takes in an object as a parameter and returns the <i>String</i> equivalent of the specified parameter.



The *String* Class: Example

```
1 class StringDemo {
2     public static void main(String args[]) {
3         String name = "Jonathan";
4         System.out.println("name: " + name);
5         System.out.println("3rd character of name: " +
6             name.charAt(2));
7         /* character that first appears alphabetically
8            has lower unicode value */
9         System.out.println("Jonathan compared to Solomon: "
10             + name.compareTo("Solomon"));
11         System.out.println("Solomon compared to Jonathan: "
12             + "Solomon".compareTo("Jonathan"));
13 //continued...
```



The *String* Class: Example

```
14      /* 'J' has lower unicode value compared to 'j' */
15      System.out.println("Jonathan compared to jonathan: " +
16                          name.compareTo("jonathan"));
17      System.out.println("Jonathan compared to jonathan
18 (ignore case): " + name.compareToIgnoreCase("jonathan"));
19      System.out.println("Is Jonathan equal to Jonathan? " +
20                          name.equals("Jonathan"));
21      System.out.println("Is Jonathan equal to jonathan? " +
22                          name.equals("jonathan"));
23      System.out.println("Is Jonathan equal to jonathan
24 (ignore case)? " + name.equalsIgnoreCase("jonathan"));
25 //continued...
```



The *String* Class: Example

```
26 char charArr[] = "Hi XX".toCharArray();
27 /* Need to add 1 to the endSrc index of getChars */
28 "Jonathan".getChars(0, 2, charArr, 3);
29 System.out.print("getChars method: ");
30 System.out.println(charArr);
31 System.out.println("Length of name: " +
32                     name.length());
33 System.out.println("Replace a's with e's in name: " +
34                     name.replace('a', 'e'));
35 /* Need to add 1 to the endIndex parameter of
36    substring*/
37 System.out.println("A substring of name: " +
38                     name.substring(0, 2));
```

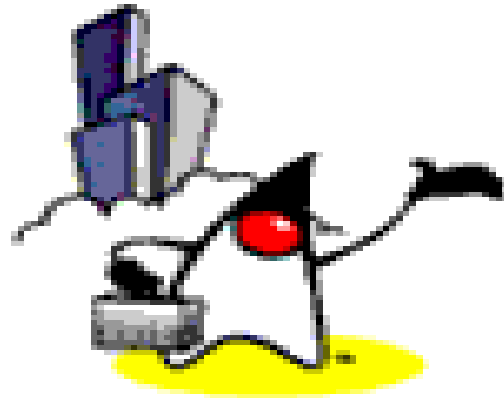
```
39 //continued...
```



The *String* Class: Example

```
40 System.out.println("Trim \"  a b c d e f  \": \"\" +  
41     \"  a b c d e f  \".trim() + "\"");  
42 System.out.println("String representation of boolean  
43     expression 10>10: \" + String.valueOf(10>10));  
44 /* toString method is implicitly called in the println  
45     method*/  
46 System.out.println("String representation of boolean  
47     expression 10<10: \" + (10<10));  
48 /* Note there's no change in the String object name  
49     even after applying all these methods. */  
50 System.out.println("name: \" + name);  
51 }  
52 }
```





StringBuffer Class

The *StringBuffer* Class

- Problem with *String* objects:
 - Once created, can no longer be modified (It is a final class)
- A *StringBuffer* object
 - Similar to a *String* object
 - But, mutable or can be modified
 - Unlike *String* in this aspect
 - Length and content may changed through some method calls



The *StringBuffer* Class: Methods

StringBuffer Methods

```
public int capacity()
```

Returns the current capacity of this *StringBuffer* object.

```
public StringBuffer append(-)
```

Appends the string representation of the argument to this *StringBuffer* object. Takes in a single parameter which may be of these data types: *boolean*, *char*, *char []*, *double*, *float*, *int*, *long*, *Object*, *String* and *StringBuffer*. Still has another overloaded version.

```
public char charAt(int index)
```

Returns the character located in the specified *index*.

```
public void getChars(int srcBegin, int srcEnd, char[] dst, int dstBegin)
```

Gets the characters from this object starting at the *srcBegin* index up to the *srcEnd* index and copies these characters to the *dst* array starting at the *dstBegin* index.

```
public StringBuffer delete(int start, int end)
```

Deletes the characters within the specified range.

```
public StringBuffer insert(int offset, -)
```

Inserts the string representation of the second argument at the specified offset. An overloaded method. Possible data types for the second argument: *boolean*, *char*, *char []*, *double*, *float*, *int*, *long*, *Object* and *String*. Still has another overloaded version.



The *StringBuffer* Class: Example

```
1 class StringBufferDemo {
2     public static void main(String args[]) {
3         StringBuffer sb = new StringBuffer("Jonathan");
4         System.out.println("sb = " + sb);
5         /* initial capacity is 16 */
6         System.out.println("capacity of sb: "+sb.capacity());
7         System.out.println("append \'O\' to sb: " +
8                             sb.append('O'));
9         System.out.println("sb = " + sb);
10        System.out.println("3rd character of sb: " +
11                            sb.charAt(2));
```

12 //continued...



The *StringBuffer* Class: Example

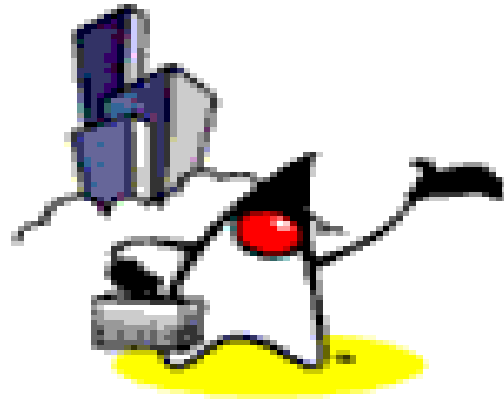
```
13      char charArr[] = "Hi XX".toCharArray();
14      /* Need to add 1 to the endSrc index of getChars */
15      sb.getChars(0, 2, charArr, 3);
16      System.out.print("getChars method: ");
17      System.out.println(charArr);
18      System.out.println("Insert \'jo\' at the 3rd cell: "
19                          + sb.insert(2, "jo"));
20      System.out.println("Delete \'jo\' at the 3rd cell: "
21                          + sb.delete(2,4));
22      System.out.println("length of sb: " + sb.length());
23  //continued
```



The *StringBuffer* Class: Example

```
24      System.out.println("replace: " +
25                          sb.replace(3, 9, " Ong"));
26      /* Need to add 1 to the endIndex parameter of
27         substring*/
28      System.out.println("substring (1st two characters): "
29                          + sb.substring(0, 3));
30      System.out.println("implicit toString(): " + sb);
31  }
32 }
```





Wrapper Class

The Wrapper Classes

- Some Facts:
 - Primitive data types are not objects
 - Cannot access methods of the *Object* class
 - Only actual objects can access methods of the *Object* class
 - Why wrapper classes?
 - Need an object representation for the primitive type variables to use Java built-in methods
- Definition:
 - Object representations of simple non-object variables



The Wrapper Classes

- Wrapper classes are very similar to their primitive equivalents.
 - Capitalized
 - Spelled out versions of the primitive data types

The Wrapper Classes: Boolean Example

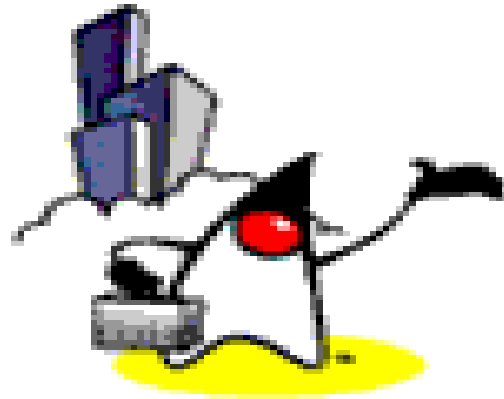
```
1 class BooleanWrapper {
2     public static void main(String args[]) {
3         boolean booleanVar = 1>2;
4         Boolean booleanObj = new Boolean("True");
5         /* primitive to object; can also use valueOf
6            method */
7         Boolean booleanObj2 = new Boolean(booleanVar);
8         System.out.println("booleanVar = " + booleanVar);
9         System.out.println("booleanObj = " + booleanObj);
10        System.out.println("booleanObj2 = " +
11                               booleanObj2);
12    //continued...
```



The Wrapper Classes: Boolean Example

```
13      System.out.println("compare 2 wrapper objects: "
14                          + booleanObj.equals(booleanObj2));
15      /* object to primitive */
16      booleanVar = booleanObj.booleanValue();
17      System.out.println("booleanVar = " + booleanVar);
18  }
19 }
```





Process Class

The *Process* Class

- Definition:
 - Provides methods for manipulating processes
 - Killing the process
 - Running the process
 - Checking the status of the process
 - Represents running programs

Process Methods
<code>public abstract void destroy()</code>
Kills the current process.
<code>public abstract int waitFor() throws InterruptedException</code>
Does not exit until this process terminates.

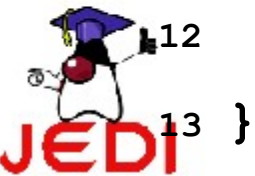
The *Runtime* Class

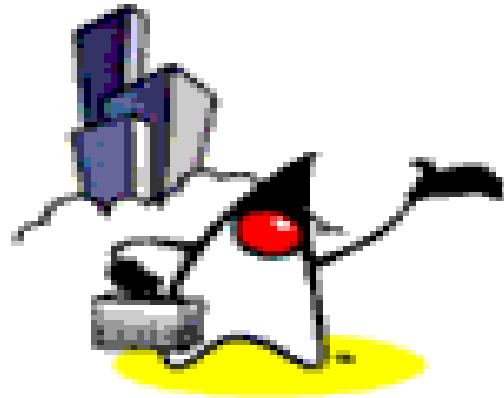
- Represents the runtime environment
- Has two important methods:

Runtime Methods
<code>public static Runtime getRuntime()</code>
Returns the runtime environment associated with the current Java application.
<code>public Process exec(String command) throws IOException</code>
Causes the specified <i>command</i> to be executed. Allows you to execute new processes.

The *Process* and *Runtime* Class: Example

```
1 class RuntimeDemo {
2     public static void main(String args[]) {
3         Runtime rt = Runtime.getRuntime();
4         Process proc;
5         try {
6             proc = rt.exec("regedit");
7             proc.waitFor();    //try removing this line
8         } catch (Exception e) {
9             System.out.println("regedit is an unknown
10                                command.");
11         }
12     }
13 }
```





System Class

The *System* Class

- Provides many useful fields and methods
 - Standard input
 - Standard output
 - Utility method for fast copying of a part of an array

The *System* Class: Methods

System Methods

```
public static void arraycopy(Object src, int srcPos, Object dest, int destPos, int length)
```

Copies *length* items from the source array *src* starting at *srcPos* to *dest* starting at index *destPos*. Faster than manually programming the code for this yourself.

```
public static long currentTimeMillis()
```

Returns the difference between the current time and January 1, 1970 UTC. Time returned is measured in milliseconds.

```
public static void exit(int status)
```

Kills the Java Virtual Machine (JVM) running currently. A non-zero value for status by convention indicates an abnormal exit.

```
public static void gc()
```

Runs the garbage collector, which reclaims unused memory space for recycling.

```
public static void setIn(InputStream in)
```

Changes the stream associated with *System.in*, which by default refers to the keyboard.

```
public static void setOut(PrintStream out)
```

Changes the stream associated with *System.out*, which by default refers to the console.



The *System* Class: Example

```
1 import java.io.*;
2 class SystemDemo {
3     public static void main(String args[])
4         throws IOException {
5         int arr1[] = new int[1050000];
6         int arr2[] = new int[1050000];
7         long startTime, endTime;
8         /* initialize arr1 */
9         for (int i = 0; i < arr1.length; i++) {
10             arr1[i] = i + 1;
11         }
12 //continued...
```



The *System* Class: Example

```
13      /* copying manually */
14      startTime = System.currentTimeMillis();
15      for (int i = 0; i < arr1.length; i++) {
16          arr2[i] = arr1[i];
17      }
18      endTime = System.currentTimeMillis();
19      System.out.println("Time for manual copy: " +
20                          (endTime-startTime) + " ms.");
21  //continued...
```



The *System* Class: Example

```
22      /* using the copy utility provided by java */
23      startTime = System.currentTimeMillis();
24      System.arraycopy(arr1, 0, arr2, 0, arr1.length);
25      endTime = System.currentTimeMillis();
26      System.out.println("Time for manual copy: " +
27                          (endTime-startTime) + " ms.");
28      System.gc();    //force garbage collector to work
29      System.setIn(new FileInputStream("temp.txt"));
30      System.exit(0);
31  }
32 }
```



System Properties

- `java.version` Java Runtime Environment version
- `java.vendor` Java Runtime Environment vendor
- `java.vendor.url` Java vendor URL
- `java.home` Java installation directory
- `java.vm.specification.version` Java Virtual Machine specification version
- `java.vm.specification.vendor` Java Virtual Machine specification vendor
- `java.vm.specification.name` Java Virtual Machine specification name
- `java.vm.version` Java Virtual Machine implementation version
- `java.vm.vendor` Java Virtual Machine implementation vendor
- `java.vm.name` Java Virtual Machine implementation name
- `java.specification.version` Java Runtime Environment specification version
- `java.specification.vendor` Java Runtime Environment specification vendor
- `java.specification.name` Java Runtime Environment specification name



System Properties

- `java.class.version` Java class format version number
- `java.class.path` Java class path
- `java.library.path` List of paths to search when loading libraries
- `java.io.tmpdir` Default temp file path
- `java.compiler` Name of JIT compiler to use
- `java.ext.dirs` Path of extension directory or directories
- `os.name` Operating system name
- `os.arch` Operating system architecture
- `os.version` Operating system version
- `file.separator` File separator ("/" on UNIX)
- `path.separator` Path separator (":" on UNIX)
- `line.separator` Line separator ("\n" on UNIX)
- `user.name` User's account name
- `user.home` User's home directory
- `user.dir` User's current working directory



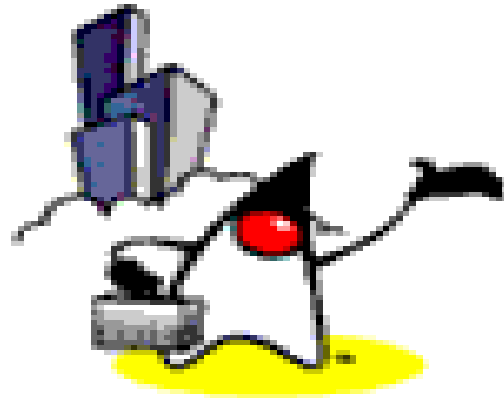
Example: Display System Properties

```
public static void main(String[] args) {  
    Properties p1 = System.getProperties();  
    p1.list(System.out);  
}
```

Example: Display System Properties

```
java.runtime.name=Java(TM) 2 Runtime Environment, Stand...  
sun.boot.library.path=C:\Program Files\Java\jdk1.5.0_06\jre...  
java.vm.version=1.5.0_06-b05  
java.vm.vendor=Sun Microsystems Inc.  
java.vendor.url=http://java.sun.com/  
path.separator=;  
java.vm.name=Java HotSpot(TM) Client VM  
file.encoding.pkg=sun.io  
user.country=US  
sun.os.patch.level=Service Pack 2  
java.vm.specification.name=Java Virtual Machine Specification  
user.dir=C:\handson2\development\javalang\samp...  
java.runtime.version=1.5.0_06-b05  
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment  
java.endorsed.dirs=C:\Program Files\Java\jdk1.5.0_06\jre...  
...
```





Properties Class

Properties Class

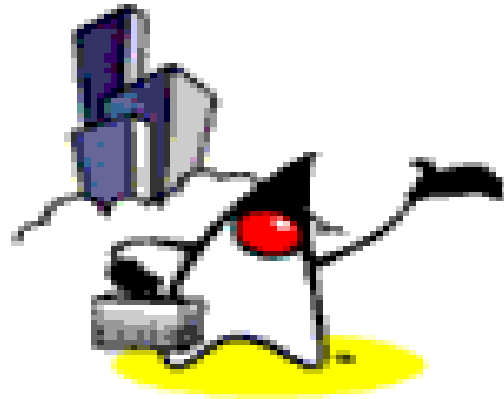
- The Properties class represents a persistent set of properties
- The Properties can be saved to a stream or loaded from a stream
 - Typically a file
- Each key and its corresponding value in the property list is a string
- A property list can contain another property list as its "defaults"; this second property list is searched if the property key is not found in the original property list



The *Properties* Class: Example

```
22      // set up new properties object
23      // from file "myProperties.txt"
24      FileInputStream propFile
25          = new FileInputStream("myProperties.txt") ;
26      Properties p
27          = new Properties(System.getProperties()) ;
28      p.load(propFile) ;
29
30      // set the system properties
31      System.setProperties(p) ;
32
33      // display new properties
34      System.getProperties().list(System.out) ;
```





Date Class

Date Class

- Represents a precise moment in time, down to the millisecond
- Dates are represented as a *long* type that counts the number of milliseconds since midnight, January 1, 1970, Greenwich Mean Time

The *Date* Class: Example

```
22 // Return the number of milliseconds in the Date
23 // as a long, using the getTime() method
24 Date d1 = new Date();
25 // timed code goes here
26 for (int i=0; i<1000000; i++) { int j = i;}
27 Date d2 = new Date();
28 long elapsed_time = d2.getTime() - d1.getTime();
29 System.out.println("That took " + elapsed_time
30                    + " milliseconds");
```





Thank You!

