Handling Strings

As noted previously in this course, Java handles character strings as real objects, not just arrays of characters.

To find help on: String class

Follow these links: **All Packages**
java.lang
String

This example shows some of the ways you can handle strings in Java:

```
// Create some new strings
String s0 = new String();              // Empty string object
String s1 = new String("Hello");       // Create from literal
String s2 = new String("World");
String s3 = "Hello World";             // Create from literal

// Join some strings together
s0 = s1.concat(s2);                 // s0 == "HelloWorld"
s1 = s1.concat(" " + s2);           // s1 == "Hello World"

// Convert case
s0 = s1.toUpperCase();              // s0 == "HELLO WORLD"
s1 = s1.toLowerCase();              // s1 == "hello world"
s3 = "Hi Mom!".toLowerCase();       // s3 == "hi mom!"

// Compare strings
boolean b1, b2;
b1 = s0.equals(s1);      // False: case sensitive
b2 = s0.equalsIgnoreCase(s1); // True: case insensitive

// Look for substrings
int n1, n2;
n1 = s0.indexOf("WORLD");           // n1 == 6
n2 = s0.indexOf("world");           // n2 == -1, not found
```

.substring(ini, fin);

Note that in filling the **Date** array, you must use the **Date( )** constructor again. The following will give a null pointer error:

```
Date jan96[] = new Date[31];
int i;

// Load second array with dates of January 1996
for(i = 0; i < jan96.length; i++) {
        jan96[i].setYear(96);          // CRASH!!!
        jan96[i].setMonth(0);
        jan96[i].setDate(i+1);
}
```

Multi-Dimensional Arrays

In Java, multi-dimensional arrays are arrays of arrays. For example:

```
int tens[][] = new int[100][100];

// tens.length is total number of elements, in this case
// 100 * 100 or 10000
for(i = 0; i < (tens.length / tens[0].length); i++)
        for(j = 0; j < tens[0].length; j++)
           tens[i][j] = (i+1)*(j+1)*10;
```

Creating Arrays from Literals

You can also create arrays from literals, such as that shown below:

```
int nums[] = {0, 1, 2, 3};
String names[] = {"zero", "one", "two", "three"};

String stonesInfo[][] = {
  {"Keith",    "Richards",     "Guitar"},
  {"Mick",    "Jagger",  "Vocals"},
  {"Ron",  "Wood",   "Guitar"},
  {"Charlie",   "Watts",      "Drums"}
};
```

Java Packages

What is a Package?

Java lets you organize your classes into groupings called *packages*:

- The classes in a package must reside in the same location.
- The classes in a package can be related by function, author, or any other useful criterion.
- If you do not explicitly assign a class to a package, it belongs to a default package.

Major Packages

The Java class library contains a number of predefined packages. Table 4-2 lists some of them with a sampling of the classes they contain:

Package	Description	Classes Include
java.lang	General language classes	Boolean, String, System, Thread
java.util	Additional useful general classes	Date, Hashtable, Stack
java.awt	GUI toolkit	Button, CheckBox, Label, TextArea
java.io	Input / output	File, InputStream, OutputStream
java.net	Network objects	InetAddress, Socket, URL

Table 4-2: Major Java packages

Using Packages

When you refer to a class in your Java source code, the compiler must know what package contains that class. Classes in the java.lang package are always known, but classes in other packages must be identified.

For example, the following code would generate compiler errors because the compiler does not recognize **Date** and **Stack**, two classes in **java.util**:

```
Date  d = new Date();
Stack s = new Stack();
```

In the following application, you might use a **StopWatch** object to time how long the user waits before entering input. You might also use Java's predefined **Date** class to handle a date/time value.

```
// StopWatchTestApp.java
import java.util.*;
public class StopWatchTestApp {
      public static void main (String args[]) {
          StopWatch swatch, timex; // Declare local objects
          swatch = new StopWatch(); // Initialize 1st obj.
          swatch.start();           // Start timing.

          timex = new StopWatch();  // Initialize 2nd obj.
          timex.start();            // Start timing.

          // Display a prompt to the user
          System.out.println("Enter a string:");
          System.in.read();              // Wait for input.

          swatch.stop();            // Stop timing.

          Date t;     // Declare date obj to hold time.
          t = swatch.getElapsedTime();

          // Print elapsed time.
          System.out.println("Elapsed time: " +
            t.toString() );
      }
}
```

In this case, **StopWatch** is the class; **swatch** and **timex** are instances of that class that run independently of each other.