Strings

- Strings in Java are objects

- Thus there is a String class (a large class)

- In a statement like this:
  System.out.println(“Hello World”);
  the Java compiler creates a String object from the quoted text

- Example string creation:

  String MyStr1 = new String(“Hello World”);

- String objects can be used anywhere a quoted string is allowed.
  for example: System.out.println(MyStr1);

Example – Declare Strings in Several Ways

```
class StringDemo {
  public static void main(String args[]) {

    String mystr1 = new String("Java strings are objects.");
    String mystr2 = "They are constructed in various ways.";
    String mystr3 = new String(mystr2);

    System.out.println(mystr1);
    System.out.println(mystr2);
    System.out.println(mystr3);
  }
}
```

Example – Declare Strings in Several Ways

```
class StringDemo {
  public static void main(String args[]) {

    String mystr1 = new String("Java strings are objects.");
    String mystr2 = "They are constructed in various ways.";
    String mystr3 = new String(mystr2);

    System.out.println(mystr1);
    System.out.println(mystr2);
    System.out.println(mystr3);
  }
}
```

```
Java strings are objects.
They are constructed in various ways.
They are constructed in various ways.
```

Operating on Strings

The **String** class contains several methods that operate on strings. Here are the general forms for a few:

boolean equals(*str*)	Returns true if the invoking string contains the same character sequence as *str*.
int length( )	Obtains the length of a string.
char charAt(*index*)	Obtains the character at the index specified by *index*.
int compareTo(*str*)	Returns less than zero if the invoking string is less than *str*, greater than zero if the invoking string is greater than *str*, and zero if the strings are equal.
int indexOf(*str*)	Searches the invoking string for the substring specified by *str*. Returns the index of the first match or −1 on failure.
int lastIndexOf(*str*)	Searches the invoking string for the substring specified by *str*. Returns the index of the last match or −1 on failure.

```
// String method operation:  length( )
class StrOps {
  public static void main(String args[]) {
    String str1 ="When it comes to Web programming, Java is #1.";
    String str2 = new String(str1);
    String str3 = "Java strings are powerful.";
    int result, idx;
    char ch;

  System.out.println("Length of str1: " + str1.length( ) );
```

```
// String method operation:  length( )
class StrOps {
  public static void main(String args[]) {
    String str1 ="When it comes to Web programming, Java is #1.";
    String str2 = new String(str1);
    String str3 = "Java strings are powerful.";
    int result, idx;
    char ch;

  System.out.println("Length of str1: " + str1.length( ) );
```

```
Length of str1: 45
```

```
                    // String method operation:   equals( )
class StrOps {
  public static void main(String args[]) {
     String str1 ="When it comes to Web programming, Java is #1.";
     String str2 = new String(str1);
     String str3 = "Java strings are powerful.";
     int result, idx;
     char ch;

  if (str1.equals(str2) )
     System.out.println("str1 equals str2");
  else
     System.out.println("str1 does not equal str2");
```

```
                    // String method operation:   equals( )
class StrOps {
  public static void main(String args[]) {
     String str1 ="When it comes to Web programming, Java is #1.";
     String str2 = new String(str1);
     String str3 = "Java strings are powerful.";
     int result, idx;
     char ch;

  if (str1.equals(str2) )
     System.out.println("str1 equals str2");
  else
     System.out.println("str1 does not equal str2");
```

```
str1 does not equal str2
```

```
                    // String method operation:   equals( )
class StrOps {
  public static void main(String args[]) {
     String str1 ="When it comes to Web programming, Java is #1.";
     String str2 = new String(str1);
     String str3 = "Java strings are powerful.";
     int result, idx;
     char ch;

  if (str1.equals(str2) )
     System.out.println("str1 equals str2");
  else
     System.out.println("str1 does not equal str2");
```

Why use method equals( ) rather than
if (str1 == str2) ??

```
                    // String method operation:  compareTo( )
class StrOps {
  public static void main(String args[]) {
    String str1 ="When it comes to Web programming, Java is #1.";
    String str2 = new String(str1);
    String str3 = "Java strings are powerful.";
    int result, idx;
    char ch;

    result = str1.compareTo(str3);
    if(result == 0)
      System.out.println("str1 and str3 are equal");
    else if(result < 0)
      System.out.println("str1 is less than str3");
    else
      System.out.println("str1 is greater than str3");
```

```
                    // String method operation:  compareTo( )
class StrOps {
  public static void main(String args[]) {
    String str1 ="When it comes to Web programming, Java is #1.";
    String str2 = new String(str1);
    String str3 = "Java strings are powerful.";
    int result, idx;
    char ch;

    result = str1.compareTo(str3);
    if(result == 0)
      System.out.println("str1 and str3 are equal");
    else if(result < 0)
      System.out.println("str1 is less than str3");
    else
      System.out.println("str1 is greater than str3");
```

```
str1 is greater than str3
```

```
// String method operation:  indexOf( )

// Create a new string

String str4 = "One Two Three One";
              0              14

idx = str4.indexOf("One");

System.out.println("Index of first occurence of One: " + idx);
```

```
// String method operation:  indexOf( )

// Create a new string

String str4 = "One Two Three One";
              0              14

idx = str4.indexOf("One");

System.out.println("Index of first occurence of One: " + idx);
```

```
Index of first occurence of One: 0
```

```
// String method operation:  lastIndexOf( )

// Create a string

String str4 = "One Two Three One";
                0              14
idx = str4.lastIndexOf("One");

System.out.println("Index of last occurence of One: " + idx);
```

```
// String method operation:  lastIndexOf( )

// Create a string

String str4 = "One Two Three One";
              0              14

idx = str4.lastIndexOf("One");

System.out.println("Index of last occurence of One: " + idx);
```

Index of last occurence of One: 14

Strings are Immutable

Once created, the character sequence that makes up the string cannot be changed.

But that is not a problem.

New strings can be created from old.

Also, another class is StringBuffer
(a topic for another day)

Strings can be concatenated together to form a new string

```
String str1 = “One”;
String str2 = “Two”;
String str3 = “Three”;

String str4 = str1 + str2 + str3;

System.out.println(str4);
```

Strings can be concatenated together
to form a new string

```
String str1 = “One”;
String str2 = “Two”;
String str3 = “Three”;

String str4 = str1 + str2 + str3;

System.out.println(str4);
```

OneTwoThree

The substring method

substring(int *startIndex*, int *stopIndex*)

```
class SubStr {
   public static void main(String args[]) {
      String origstr = "Java makes the Web move.";

            // construct a substring
      String substr = origstr.substring(5, 18);

      System.out.println("origstr: " + origstr);
      System.out.println("substr: " + substr);
   }
}
```

The substring method

substring(int *startIndex*, int *stopIndex*)

```
class SubStr {
   public static void main(String args[]) {
      String origstr = "Java makes the Web move.";

            // construct a substring
      String substr = origstr.substring(5, 18);

      System.out.println("origstr: " + origstr);
      System.out.println("substr: " + substr);
   }
}
```

```
origstr: Java makes the Web move.
substr: makes the Web
```

A string can be used to control a switch statement

```
String command = “cancel”;

switch(command) {
    case “connect”:
        System.out.println(“Connecting”);
        break;
    case “cancel”:
        System.out.println(“Canceling”);
        break;
    default:
        System.out.println(“Command Error”);
        break;
}
```

For-Each Style Loops

Useful when the elements of an array are sequentially accessed.

Syntax definition:

for(*type iteration-var : collection*) *statement-block*

Example:

```
        // first declare variables
int  data[ ] = {11, 12, 13, 14, 15, 16, 17, 18};
int sum;
        // example loop using traditional method
for (int i=0; I < 8; i++) sum += data[i];
```

For-Each Style Loops

Cycles through a collection of items sequentially.
Useful when the elements of an array are sequentially accessed.

Syntax definition:

for(*type iteration-var : collection*) *statement-block*

For-Each Style Loops

Cycles through a collection of items sequentially.
Useful when the elements of an array are sequentially accessed.

Syntax definition:

for(*type iteration-var : collection*) *statement-block*

Examples:

```
        // first declare variables
int  data[ ] = {11, 12, 13, 14, 15, 16, 17, 18};
int sum;
        // example loop using traditional method
for (int i=0; I < 8; i++) sum += data[i];
```

For-Each Style Loops

Cycles through a collection of items sequentially.
Useful when the elements of an array are sequentially accessed.

Syntax definition:

for(*type iteration-var : collection*) *statement-block*

Examples:

```
        // first declare variables
int  data[ ] = {11, 12, 13, 14, 15, 16, 17, 18};
int sum;
        // example loop using traditional method
for (int i=0; I < 8; i++) sum += data[i];

        // example using For-Each style
for (int x: data) sum += x;
```