Object Oriented Programming with Java

Strings

Character Strings

- A string of characters can be represented as a **string literal** by putting double quotes around the text.

  Examples:

```
"This is a string literal."
"123 Main Street"
"X"
```

- Character strings are objects in Java, defined by the `String` class. This means you can call methods on a string.

Strings and Java

- A **`String`** in Java can be created in two ways:
  - By calling the **String** constructor with **new**
  - By creating a **String** literal

```
String a = new String("GMU");

String b = "GMU";
String c = "GMU"
String d = "G"+"MU";
```

- String literals are ***interned*** in the **string pool** while the ones created with **new** are not. What does this entail?

```
if(a == b)   // false
    System.out.print("not really");
else if(c == b)  // true
    System.out.print("yeah!");
else if(d == b)  // true
    System.out.print("yeah!");
```

Comparing Strings

- String comparison with the `==` operator is based on memory addresses because strings are objects (i.e. reference type).
- For **content comparison** you must use the `.equals` method
- The `equals` method determines if two Strings contain exactly the same characters in exactly the same order.
- The `equals` methods returns a boolean value.

```
String name1 = "GMU";
String name2 = "gmu";
if (name1.equals(name2))   // false
    System.out.println("case insensitive");
```

Useful String methods

```
String s = "literal value"
String s = new String(anotherString)
String s = new String(char_array[])
s.length()
s.charAt(index)
s.getChars(start, end, char_buffer[])
s.equals(anotherString)
s.equalsIgnoreCase(anotherString)
s.compareTo(anotherString)
s.startsWith(anotherString)
s.endsWith(anotherString)
```

```
s.regionMatches
s.indexOf(anotherString)
s.lastIndexOf(anotherString)
s.substring(start, length)
s.concat(anotherString)     // just like + operator
s.toCharArray()
s.trim()
s.toUpperCase()
s.toLowerCase()
s.replace(str1, str2)
s.split()// returns array of Strings
```

Escape Sequences

- The following confuses the compiler: it interprets the second quote as the end of the string `"I said "`.

```
System.out.println ("I said "Hello", Bob."); //ERROR
```

- *escape sequence:* series of characters that represents a special character
  → begins with a backslash character \

```
System.out.println ("I said \"Hello\", Bob."); //OK
```

Escape Sequences

Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

Escape Sequence: Example

```
System.out.println ("Roses are red,\n\tViolets are blue,\n" +
        "Sugar is sweet,\n\tBut I have \"commitment issues\",\n\t" +
        "So I'd rather just be friends\n\tAt this point in our " +
        "relationsha\bip.");
```

output:

```
Roses are red,
    Violets are blue,
Sugar is sweet,
    But I have "commitment issues",
    So I'd rather just be friends
    At this point in our relationship.
```

String Concatenation

- The *string concatenation operator* (+) is used to append one string to the end of another

```
"Peanut butter " + "and jelly"
```

- It can also be used to append a number to a string
  → The + operator is also used for arithmetic addition
  → operand types dictate which meaning + has.
- A string literal cannot be broken across two lines in a program

String Concatenation

- If at least one of the operands are strings, it performs string concatenation (by converting the other one to a String if necessary)
- The + operator is evaluated left to right, but parentheses can force the order

```
int x=7, y=12;
int total = x+y;                    // Does addition
String myString = "George";
String name;
name = myString + " Mason";    // Does String concat
```