CMSC131

Characters, Strings

Scanner Methods

Characters versus Strings

In Java:

- "S" is a **String** object of length 1
- 'S' is a **char** primitive (characters are always "length" of 1)

This "strong typing" and distinction between the two differs from some other languages (like Python), and we will explore the reasons now and as the semester continues.

The **String** type

The **String** data type is not a primitive type in Java, is what we call a **Class** and when we declare one to use it is called a **reference** to an **object**.

- We will discuss these ideas more later, but for now imagine that a variable holding a ***reference*** to a **String** object knows where the actual **String** object lives in memory. That object will store information.
- A **String** object is what's known as ***immutable***; once it is created it can be read but cannot be altered.

The **String** type

As we saw earlier, you can concatenate two **String** objects using the **+** operator but the **+** operator can also be used for math operations.

The left operand "decides" which form of **+** is being used.

Some Special Character Values

- single quote: \'
- double quote: \"
- new line (cr/lf): \n
- tab: \t
- single backslash: \\

Creating **String** objects

There are two ways to create String objects in Java; implicitly and explicitly.

- If you surround keyboard characters with " " Java will automatically create a String object using the contents. This means that it is valid to do:

```
String myStr = "Hello World!";
```

- However, in general, when you want a new object in java you ask for a new one to be created, and if there is some initial information you want stored, you specify it and something called the class' constructor is used to initialize the contents:

```
String myStr = new String("Hello World!");
```

SomeStrings.java example: Part 1

```
public class SomeStrings {
  public static void main(String[] args) {
      String name1 = "Sam";
      String name2 = "Pat";
      System.out.println(name1+" "+name2);

      System.out.println("Hi\nThere\tClass\\\\\n\n");}
}
```

SomeStrings.java example: Part 2

```
public class SomeStrings {
  public static void main(String[] args) {
      String name1 = "Sam";
      String name2 = "Pat";
      char letter = 'A';
      int number1 = 14;
      float number2 = 17.5F;

      System.out.println(name1+letter+number1+number2);
      System.out.println(name1+letter+(number1+number2));
  }
}
```

Will the same thing be printed twice in a row by the above or will the two print statements create different output than each other?

Will the output be…

0% A. The Same

0% B. Different

```
String name1 = "Sam";
String name2 = "Pat";
char letter = 'A';
int number1 = 14;
float number2 = 17.5F;

System.out.println(name1+letter+number1+number2);
System.out.println(name1+letter+(number1+number2));
```

Response Counter

Fastest Responders

Seconds	Participant	Seconds	Participant

Strings and Methods

In object-oriented languages, a class can be used to describe a structure to hold information (objects) as well as operations that can be performed (methods) and will often serve a dual-purpose of doing both (describing the structure of the object and the operations that can be performed on them).

- We looked at primitive values and are now looking at the String class provided by Java. For some things Java allows operators to be used with this class (such as `+`) but in other situations, we will need to make use of a ***method*** from the class.

Instance Methods

A **method** is basically a way of expressing that a block of code can be called and executed which also provides a way for information to be given to that block of code as well.

An **instance method** is one that is invoked using a particular object, and that therefore has access to the information within that particular object.

When we have called `System.out.println()` we've been having an object `System.out` invoke the method `println`. That method knows the object that invoked it and is able to make use of it. The net result in that example is the printed text is displayed "to" the output stream that `System.out` refers to (rather than to somewhere else).

String Comparison

String objects are compared using **Methods** rather than **Operators**.

string1.equals(string2) will return a boolean value

string1.compareTo(string2) will return an integer value using the following rules based on the lexicographical order of the strings:

- If the result is less than 0 string1 precedes string2
- If the result is equal to 0 string1 matches string2
- If the result is greater than 0 string1 succeeds string2

NOTE the negative value doesn't have to be -1 and the positive value doesn't have to be +1

Uppercase/Lowercase Conversions

The String class provides instance methods named `toUpperCase()` and `toLowerCase()` that do almost what they sound like.

– REMINDER: In Java, once you create a **String** object, the contents are not allowed to be changed.

If you have an existing String object and use it to invoke `toUpperCase()` then that method will ***return a reference to a brand new String object*** which contains the uppercase version of the original one.

Upper.java example

```
public class Upper {
  public static void main(String[] args) {
     String name = "Sam";

     name.toUpperCase();
     System.out.println(name);

     String uName = name.toUpperCase();
     System.out.println(uName);
  }
}
```

What will be printed by each of the above?

Static Methods

A static method is one that is invoked using the name of a class rather than a specific object reference, and that therefore has no access to information held within particular objects.

As an example in the String class, it provides a static method called `valueOf(float f)` that takes a floating point value as a parameter and returns an object holding a character-based representation of that numeric value. This could appears as:

```
String result = String.valueOf(3.141592);
```

The variable **result** stores the object reference returned by the static method. No existing object was used or needed.

Input using the Scanner class

The **Scanner** class is included as part of the utility libraries available in Java 5.0 and later but we need to **import** it to make use of it in a Java program.

```
import java.util.Scanner;
```

It allows us to obtain data from an input source (such as a keyboard) and also "convert" that data into the format of different Java types via instance methods.

Creating and Using

We will need to declare and create a Scanner object:

```
Scanner myScanner = new Scanner(System.in);
```

We can then use any of the **Scanner** class methods to read (and "convert" data) such as…

- **`nextBoolean()`**
- **`nextFloat()`**
- **`nextInt()`**
- **`next()`**

```
//reads/returns characters until next whitespace
```

- **`nextLine()`**

```
//reads/returns characters until the end of the input line
```

SimpleInput.java example

```
import java.util.Scanner;
public class SimpleInput {
  public static void main(String[] args) {
     Scanner myScanner = new Scanner(System.in);
     int i;
     float f;
     String s;

        i = myScanner.nextInt();
        System.out.println("The integer was a " + i);

        s = myScanner.next();
        System.out.println("The next \'word\' was " + s);

        s = myScanner.nextLine();
        System.out.println("Rest of the line was " + s);
  }
}
```