CS620
Structured Programming
Introduction to Java

Day 6 - Lecture 2

String Class

Strings

- A String is a sequence of characters (digits, letters etc). For example:
  - “goal”
  - “What a goal”
  - “Can I have a cup of tea”
  - “Strings can be made up of numbers”
  - “123.89f”
  - “a:\\Java\\Hello.java”

You’ve seen Strings before in S.O.P statements

String Creation

There are several ways to create a String

```
String s;
s= “Hello World”  ;

String s = “Hello World” ;

String s = new String(“Hello World” );
```

Strings

- In Java, ‘String’ is a class that we can use to create String objects

- When examined we can find that Strings are actually just arrays of chars
  - Though there’s more to Strings than just that, since String is a class

Strings

- We said before the arrays have a fixed length once created
- From Oracle:
  - The String class has a number of methods that appear to modify strings.
  - What these methods really do is create and return a new string that contains the result of the operation.

String Length & Characters

- When we created publically accessible methods in our Bicycle class previously to return the values of our private variables, we were creating *accessor* methods

- The String class has accessor methods such as *length():*
  - `int len = someString.length(); // Get string length`

String Palindrome Example

- A *palindrome* is a word or sentence that is symmetric—it is spelled the same forward and backward, ignoring case and punctuation.

- *StringDemoPalindrome.java* is a short and inefficient program to reverse a palindrome string.

- It invokes the String method charAt(i), which returns the ‘i’th character in the string, counting from 0.

String Concatenation

- Method for concatenating (joining) strings:
  - `string1.concat(string2);`
    - This returns a new string containing string1 and string 2 glued together
  - `string1.concat(“Hello”);`
    - The same can also be done with string literals
- Or more simply, as we’ve seen before:
  - Use the ‘+’ operator
    - `string3 = string1 + string2;`

String Equality

- If you write:
  - `if string1 == string2 then …`
  - Then your program will check if string1 and string2 are the same object
    - i.e.; if their **references** point to the same place in memory
- This is not the same as checking if they are the same string!

String Comparison Methods

- Use the following methods for string comparisons instead:

- String.compareTo(String anotherString)
  - Examples:
    - string1.compareTo(“Hello”)
    - string2.compareTo(string1);
  - Returns 0 if the strings are equal

String Comparison Methods

- .compareToIgnoreCase()
  - The same as compareTo, but without case-sensitivity

- .equals()
  - Check if two strings are the same

- .equalsIgnoreCase()
  - As above, but without case sensitivity

String methods 1

- Let’s start by creating two String instances

```
String s1 = new String(“This is my first String”);
String s2 = new String(“Another String”);
```

- Gives the length of a String as an <int>

```
<String_1>.length() // returns an int
```

- Change String to uppercase or lowercase

```
<String_1>.toUpperCase()
<String_1>.toLowerCase()
```

- See what char is at position <int>

```
<String_1>.charAt(<int>)
```

String methods 2

- Returns the index within this string of the first occurrence of the specified character

```
<String_1>.indexOf(<char>)
```

- Replaces the first char with the second character specified

```
<String_1>.replace(<char>, <char>)
```

Strings method 3

- To generate a substring of a String(specify start and end position as ints)

```
<String_1>.substring(<int>, <int>)
```

- To generate a substring of a String(specify start position and always ends at end of String)

```
<String_1>.substring(<int>)
```

- Also, you can concatenate two Strings using the + sign e.g

```
String s  = hello;
String s1 = s+“world”;
String s2 = s + s1;
```

Or you can use `<String_1>.concat(<String_2>)`

Strings methods 4

Test if 2 Strings are lexicographically equal

```
<String_1>.equals(<String_2>)  // returns a boolean
<String_1>.compareTo(<String_2>)  // returns an int

if(s1.equals(s2)==true){}

if(s1.compareTo(s2)==0){}
```

Can also use `.equalsIgnoreCase()` and

`.compareToIgnoreCase()`

Strings comparisons

Remember Strings are objects and not primitive types.

You can’t compare Strings using <, >, <=, >=.

When we compare Strings using ==, we are not comparing contents but references. E.g.,

```
String s = new String(“Hello”);
String s1 = new String(“Hello”);
String s2 = new String(s1);
```

The expression s1 == s2 evaluates to true.
But the expression s == s1 evaluates to false because s and s1 do not refer to the same object.

As a rule of thumb, use the .equals() or .compareTo() methods to see if two Strings are lexicograpically equal.

String Class Documentation

- JavaDoc:
- http://docs.oracle.com/javase/7/docs/api/java/lang/String.html

String Examples

- CharArrayToString.java
- StringContains.java
- StringDemoPalindrome.java
- StringReplace
- StringSplit
- StringTrim