Continuing Data Types, Strings

Ryan Nixon

Objects, in Brief

- Classes, when instantiated, become instances of that class called objects. There may only be one class per running program and as many instances of that class as there is available memory

- Each class & instance has attributes (variables) to describe it and methods (actions) that it can perform

- Objects are the foundation of Java. Almost everything that you use that isn't a primitive is likely an object

String Objects

- Strings are very simple objects, but not primitives. You can easily note the difference by how it is declared:

```
String text = "Sample text";
```

- Note the capital S in String, versus lowercase letters for the primitives such as int and double

- As you'll see, because strings have been implemented as objects they gain all of the features that objects are capable of, such as methods

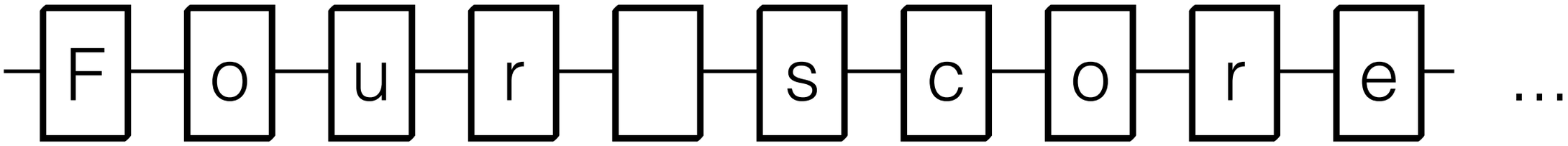
Strings as a Data Type

- A string is actually very simple to visualize. It stores a list of chars (called an array), ‘strung’ together in sequence

- Java then has shortcuts for working with these characters. It provides many useful methods to manipulate them and reserves the double quotation marks " to represent strings

- Using “ to make String so accessible allows String objects to be treated almost like primitives in your code vs. the additional steps most other objects require.

Strings as a Data Type




```
String str = “Four”;
System.out.println(str);
```

is similar in concept to:

```
char ch1, ch2, ch3, ch4;
ch1 = ‘F’;
ch2 = ‘o’;
ch3 = ‘u’;
ch4 = ‘r’;
System.out.println(ch1 + ch2 + ch3 + ch4);
```

Because of the way chars are handled in Java though you will actually get a completely different output

String Concatenation

- Assigning a string to a variable is just like any other variable. It also allows reassignments and combinations:

```
String timeRange = “seven years”;
String withTense = timeRange + “ ago”;
String punctuated = “four score and ” + text + “.”;
```

- The + operator here performs what is called string concatenation, combining all of the variables together. Note that you need to mind your spaces when concatenating, otherwise you might end up with “four score andseven yearsago.”

- You can also mix in other variables with these strings. As long as at least one variable in the chain is a String then Java will attempt to store the result as a String:

```
String numericized = 4 + “ score and “ + 7 “ years ago.”;
```

Escape Characters

- What if you wanted to quote Lincoln?
  `String quote = “”Four score and seven years ago.””;`

- This wouldn’t work well. The second “ on the line would close out the string and the remaining `Four score and...` text would be considered Java code.

- To get around this you use the backslash `\` escape character:
  `String quote = “\”Four score and seven years ago.\””;`

- The escape character signals Java that the next character in the sequence should be treated specially. It allows the double quotations to appear in a string. It also allows newlines to be easily used with `\n`, or single quotation chars to be set with `\’`:
  `char quotation = ‘\’’;`

- The backslash is not considered a real character and won’t be stored in the resulting text. If you actually want a backslash you can use `\\` which will insert a single \ into the string.

String Methods

- A big reason why strings are presented as fully-defined objects are the numerous methods that go with the String class

- For example, you can call the length() method on a string which returns the number of characters in that string as an integer:

```
String shortStr = "I'm 5";
int strLength = shortStr.length(); // 5
```

Note that length() considers all characters regardless of what they are. White space (spaces, tabs, carriage returns) all count.

- Let's go over a few of these utility functions

charAt(int)

- Returns the character within the string at the specified position.
  Param 1 - The 0-indexed position in the string

- For example:

  `“#! /bin/bash”.charAt(1);`
  Returns ‘!’

equals(String)

- Returns boolean true if the current string is character-for-character the same as the provided string.
  Param 1 - The string to compare against

- This is more useful than it looks. Comparing two objects is sometimes a tricky thing

- For example:

```
“all for one”.equals(“all for 1”); //False
“all for one”.equals(“all for One); //Also false
“all for one”.equals(“all for one”); //True
```

- You can also use `.equalsIgnoreCase(String)` to allow uppercase and lowercase variants of the same letter to match

indexOf(String)

- Returns the position (0-indexed) of the first occurrence of the provided string within the current string.
  Param 1 - The String to search for

- Returns -1 if no match is found

- For example:

```
“Now in Technicolor”.indexOf(“Tech”);
```

Returns 7

toLowerCase()

- Returns the original string with all characters converted to their lowercase equivalent.

- For example:

  `“Case matters with iThis & iThat”.toLowerCase()`

  Returns “case matters with ithis & ithat”

- There is also a companion method to this, toUpperCase(), that uppercases all characters.

replace(String, String)

- Replaces all occurrences of a given string within a larger string with new text.
  Param 1 - The original text to search for
  Param 2 - The replacement for that text

- If no text is found, the original string is returned

- For example:

```
“Denial ain't just a river in Egypt.”.replace(“Denial”,
“The Nile”);
```

  Returns “The Nile ain’t just a river in Egypt.”

substring(int, [int])

- Allows a subset of a string to be extracted.
  Param 1 - The position to start at, 0-indexed
  Param 2 - The position to end at. If not provided then substring will include the remaining characters in the string

- For example:

  ```
  “Aziz, light!”.substring(6, 11);
  ```
  Returns “light”

trim()

- Trim removes all whitespace (spaces, tabs, carriage returns) from the beginning and end of a string.

- For example:

  Before: “ `Dang cat, get off the keyboard.`”

  After: “`Dang cat, get off the keyboard.`”

The Java API

- The Java API (Application Programming Interface) through Oracle provides an excellent online reference for the available classes and methods in Java.

- I would highly suggest skimming through the String class documentation to see some of the other methods that are available to you.

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

Questions?