Ollscoil na Gaillimhe
University of Galway

CT326 Programming III

LECTURE 6

STRINGS

DR ADRIAN CLEAR
SCHOOL OF COMPUTER SCIENCE

Objectives for today

- Understand java Strings and how to handle them
- Demonstrate how to create, query, and manipulate Strings in Java

String handling

- You can store alphanumeric characters within a variable using a *String* object.
- You do not use the *NULL* character to indicate the end of a Java *String* object, as you would in *C* or *C++*:
  - The *String* object itself keeps track of the number of characters it contains.
- Using the *String* class *length* method, your application can determine the number of characters a *String* object contains.

Querying Strings

- `int length()` Returns the length of the String
- `char charAt(int index)` Returns the char value at the specified index
- `int indexOf(int ch)` Returns the index within this string of the first occurrence of the specified character.
- `boolean isBlank()` Returns true if the string is empty or contains only white space codepoints, otherwise false.
- `boolean startsWith(String prefix)` Tests if this string starts with the specified prefix.
- `boolean matches(String regex)` Tells whether or not this string matches the given regular expression.

Creating Strings

- `String myString = "Hello world!";`
- `String myString = new String();`
- `char[] ct326Array = { 'C', 'T', '3', '2', '6' };`
  `String ct326String = new String(ct326Array);`

- Formatting Strings

```
String fs;
fs = String.format("The value of the float variable is %f, while the
value of the integer variable is %d, and the string is %s", floatVar,
intVar, stringVar);
System.out.println(fs);
```

Concatenating Strings

- The (+) operator lets your application append (concatenate) one string's contents to another.
- When you concatenate a numeric value, e.g., of type *int* or *float*, Java automatically calls a special function named *toString:*
  - This converts the value into a character-string.
- To simplify the output of class-member variables, you can write a *toString* function for the classes you create:
  - In the same way as for built-in types, the *toString* function for your class will be called automatically.
- `String concat(String str)` Concatenates the specified string to the end of this string.

Comparing Strings

- `int compareTo(String anotherString)` Compares two strings lexicographically.
- `boolean equalsIgnoreCase(String anotherString)` Compares this String to another String, ignoring case considerations.

Manipulating Strings

- `String substring(int beginIndex, int endIndex)` Returns a string that is a substring of this string.
- `String [] split(String regex)` Splits this string around matches of the given regular expression.
- `String strip()` Returns a string whose value is this string, with all leading and trailing white space removed.
- `String replace(char oldChar, char newChar)` Returns a string resulting from replacing all occurrences of oldChar in this string with newChar.

In-class demo

- In the following example, the `validEmail` method takes in a String as a parameter and prints to the console whether the String is a valid format for an email address (-@-.-)
- Uses `strip()` and `toLowerCase()` to clean the String for printing
- Uses `isEmpty()` and `isBlank()` to validate it
- and uses `split()` and `length()` to test the values of the string

String handling

- String objects are immutable - that is, they cannot be changed once they've been created.
- The j*ava.lang* package provides a different class, `StringBuffer`, which you can use to create and manipulate character data on the fly.

In-class demo

- In the following example, the `reverseIt` method creates a StringBuffer, dest, the same size as source:
  - It then loops backwards over all the characters in source and appends them to dest, thereby reversing the string.
  - It finally converts dest, a StringBuffer, to a String.

Next time…

- Nested inner classes