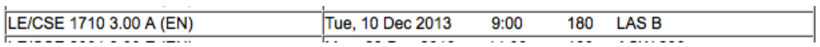
CSE1710

Week 13, Lecture 24

Fall 2013 ◆ Thursday, Dec 05, 2013

Big Picture

- LAST CLASS TODAY!!!
- Assignment is due Friday Dec 6th, 11pm (on-line submission)
- Final Exam!!



LE/CSE 1710 3.00 A (EN)	Tue, 10 Dec 2013	9:00	180	LAS B

Family Names starting with A-L
report to LAS1002 to do **LABTEST** 9:00-10:25am

Family Names starting with M-Z
report to LAS B to do **WRITTEN TEST** 9:00-10:25am

everyone switches at the half-way point (10:30am)

Revisiting the `String` class

We discussed many important methods from the String class

```
length()
charAt(int)
substring(int,int) (int)
indexOf(String), indexOf(String,int)
toString(); equals()
compareTo(String)
toUpperCase(), toLowerCase()
```

Revisiting the `String` class

We discussed many important methods from the String class

```
length()
charAt(int)
substring(int,int) (int)
indexOf(String), indexOf(String,int)
toString(); equals()
compareTo(String)
toUpperCase(), toLowerCase()
```

There is one more we will cover:

```
split(String)
```

Example

```
String str = “Here is a string!”;

String[] tokens;
// this declares a variable that has the type array
of String elements

tokens = str.split(“ ”);
// here we invoke the split method, which returns an
array of String elements, we assign the RHS to the
variable we already declared

int numTokens = tokens.length;
// here we determine the number of elements in the
array
```

5

Example

```
// here we iterate over the elements of the array
// using a for loop

for (int index = 0; index < numTokens; index++) {

      System.out.println(tokens[index]);

}
```

6

Example

```
// here we iterate over the elements of the array
// using collection-based iteration

for (String s : tokens) {

      System.out.println(s);

}
```

7

StringBuffer, a really cool class

Examples:

```
StringBuffer buf1 = new StringBuffer(“Hi”);
buf1.append(“ ”);
buf1.append(“There!”);
buf1.append(“\n”);


StringBuffer buf2 = new StringBuffer("!");
buf2.insert(0, "Hi");
buf2.insert(2, "There!!");
buf2.insert(2, " ");
buf2.append("\n");
buf2.delete(8,10);
```

8

StringBuffer, a really cool class

Examples:

```
StringBuffer buf3 = new StringBuffer("Notification");
buf3.reverse();
System.out.println(buf3.toString());
```

9

	`String`	`StringBuffer`
state? attributes?	sequence of characters	sequence of characters
object is mutable? (state can be changed)	NO!	YES!
has mutator methods?	NO!	YES! `append(String)` `delete(int, int)` `insert(int, String)` `reverse()`
objects can be operands with + operator?	YES! **can masquerade as primitive operand** can ***also*** invoke methods on object reference	NO! can ***only*** invoke methods on object reference
instantiation?	standard way `String s = new String("Hi");` also can use way that masquerades as primitive `String s = "Hi";`	***only*** the standard way `StringBuffer s;` `s = new StringBuffer("Hi");`

String instantiation: differences?

We have **two** ways to instantiate string objects…

```
String s1 = new String(“Hi”);
String s2 = “Hi”;
```

Are these two ways actually identically the same?

11

String instantiation: differences?

We have **two** ways to instantiate string objects…

```
String s1 = new String(“Hi”);
String s2 = “Hi”;
```

Are these two ways actually identically the same?

actually….
not exactly the same (next example will illustrate)

12

String instantiation via constructor

```
String s1 = new String(“Hi”);
String s2 = new String(“Hi”);
String s3 = new String(“Hi”);
```

With the regular, old “constructor” approach,
these statements will result in **the creation of 3 different `String` objects** at run time.

The objects do happen to have the same state, but they are indeed different objects.

13

String instantiation via shortcut

```
String s4 = “Hi”;
String s5 = “Hi”;
String s6 = “Hi”;
```

With the “shortcut” approach,
these statements will result in **the creation of only one `String` object** at run time.

The shortcut approach will reuse a String object if one already exists with the required state.

There is a “pool” at run time to keep track of `String` objects created via this shortcut method.

14

and while we’re at it…

Let’s take another look at Strings masquerading as primitive operands…

What is actually happening here?

```
String s1 = “X” + “Y”;
```

15

and while we’re at it…

Let’s take another look at Strings masquerading as primitive operands…

What is actually happening here?

```
String s1 = “X” + “Y”;
```

…gets transformed to…

```
String s1 =
       new StringBuffer().append(“X”).append(“Y”).toString();
```

16