CSE 1710

Lecture 19

Strings: Recap and Review of Core Concepts

RQ6.1

What is the difference between a *string object*, a *string reference*, and a *string literal*?

```
String s1 = new String("apple");
String s2 = "orange";
boolean isTheSame = s1==s2;
boolean hasTheSameState = s1.equals(s2);
boolean hasTheSameState2 = s2.equals(s1);
```

- a string *object* is an entity created at run-time, either through explicit or implicit creation
  - explicit -> the use of the String constructor
  - implicit -> the use of the String "literal"
- a string *reference* is a variable that stores an address in the JVM heap space or the special value null;
  - the address corresponds to the location of a string object
- a string *literal* is a a syntactic construct that causes a string object to be created.

True or false: anything that can be done using the `StringBuffer` class can also be done using the `String` class?

Why do we need the `StringBuffer` class?

- True (with enough additional statements and/or objects)
- the key difference is the mutability of the objects
- `StringBuffer` allows us to create mutable objects.
- `StringBuffer` provides mutators, String does not:
  - `insert(int, String)`
  - `append(String)`
  - `delete(int, int)`

What is a regular expression?

- A string, possibly consisting of special characters, that is interpreted as a pattern specification

In which contexts would a string be interpreted as a regular expression?

- as a parameter to:
  - `replaceAll(String, String)`
  - `replaceFirst(String, String)`)
  - `matches(String)`
- NOT:
  - in the string constructor
  - `indexOf(String)`, etc…

Ex 6.11

Predict the outcome of the following fragment:

L19App2

```
final int A = 7;
final int B = 4;
StringBuffer sb = new StringBuffer("University");
sb.delete(A, sb.length()).insert(A, sb.charAt(B));
output.println(sb);
```

- How to answer questions such as this one:
  - first, recognize that the fourth line can be decomposed:

    ```
    sb.delete(A, sb.length());
    sb.insert(A, sb.charAt(B));
    ```
  - second, read the API for `delete(int, int)` and `insert(int, String)`

Ex 6.13

Derive the correct REGEX:

```
final String REGEX = ?
String ss = input.nextLine();
output.println(ss.replaceAll(REGEX,"x"));
```

L19App1

E.g., if

```
ss="2456 24567: 23546:42356"
```

output:

```
"2456 x x42356"
```

- first, break the task into smaller tasks
  - match any single digit followed by an colon
  - match any multiple digit number followed by a colon

Ex 6.3a

What is the difference between an empty string and a null string?

L19App3

- an empty string is a string object
  - its state is the character sequence that consists of a 0-length sequence
- a `null` string is…
  - it refers to a string reference
  - a reference can be one of two possibilities:
    - an address at which a string object can be found
    - a reserved keyword `null`
  - a “`null` string” is a misnomer; it should be “`null` string reference” – a reference that has the value `null`

Ex 6.3b

What is wrong with the statement “A null string has zero length”?

- a null string does not have any sort of length
- length refers to the number of characters in the character sequence (a string object’s state)
- only string objects has character sequences
- a null string is actually referring to a string reference
  - the reference has a value
  - this value can be a number or null

Ex 6.22

Explain out the append method of StringBuffer works.
Could this method have been made void?

```
StringBuffer buf = new StringBuffer(“hi”);
buf.append(“ there”);
```

- the character sequence of the passed string is appended to the end of the character sequence of the object that is being mutated
- somewhat equivalent to

  ```
  String s = buf.toString() + “ there”;
  buf = new StringBuffer(s);
  ```
- Could have been void, but then we could not use the following:

  ```
  buf.append(“ you”).append(“ !”);
  ```

Ex 6.17

What is a wrapper class, and why is it needed?

- a wrapper class is a class that corresponds to a primitive type

`int`	`Integer`
`double`	`Double`
`byte`	`Byte`
`boolean`	`Boolean`

  …and so on
- it provides allows us to represent primitive values as objects
- the class definitions provide useful services (both static and non-static)
  – e.g., `Integer.parseInt(String)`

Ex 6.14

Write a program that reads a string containing two space-delimited integers from the user and outputs their sum.

E.g., given `"12 8"` the output should be `20`

L19App4a,
L19App4b

Identity the steps

1. read input
2. divide the string into the two components
3. transform each component from a string object to an int value
4. add the int values and output the sum

Strategy

– do steps 2-4 first, then step 1 last