More on Strings

Lecture 10
CGS 3416 Spring 2016

March 22, 2016
What we know so far

- In Java, a string is an object.
- The `String` class is used to create and store immutable strings.
- Some `String` class methods we have used before:
  - `equals()` – for comparing two strings (i.e. their contents), returns true or false.
  - `equalsIgnoreCase()` - just like `equals()`, except that the case of the letters doesn't matter in making a match.
  - `compareTo()` – also for comparing two strings, good for sorting.
- Don't try to compare strings by using `==`, `<`, `>`, etc. These would only compare the String reference variables, not the String objects themselves.
- Other comparison methods include `regionMatches`, `startsWith`, and `endsWith`. See String class API for full details.
Concatenation

- `concat()` - String concatenation. Returns a concatenation of two strings.

  ```java
  String s1 = "Dog";
  String s2 = "food";
  String s3 = s1.concat(s2);
  //s3 now stores "Dogfood"
  //note: s1 and s2 are NOT changed
  ```

- The `+` symbol also performs String concatenation (as we've already used in print statements).

  ```java
  String s1 = "Cat";
  String s2 = "nap";
  String s3 = s1 + s2;
  //s3 now stores "Catnap" (s1, s2 unchanged)
  ```
Substrings

- `substring()` – extracts part of a string and returns it.
- Takes in two parameters (begin index and end index) or 1 parameter (begin index).
- First character in a String has index 0. Substring returned is the index range [begin,end).
Substrings

String s1 = "Hello, World";
String s2 = s1.substring(0,5);// s2 is now "Hello".
    // picks up indices 0 - 4

String s3 = s1.substring(0,7) + "Dolly";
System.out.print(s3);// prints "Hello, Dolly"
System.out.print(s3.substring(4)); //prints "o, Dolly"

    // can even use substring on string literals
String s4= "What’s up doc?".substring(10,13);
    // s4="doc"
String length

- `length()` – returns a string’s length (number of characters).

```java
String s1 = "Hello";
String s2 = "Goodbye world";

System.out.print(s1.length()); // output: 5
System.out.print(s2.length()); // output: 13
```
charAt() method

- `charAt()` – returns a specific character, given an index.

String `s1 = "Rumplestiltskin";

System.out.print(s1.charAt(0)); // output: R
System.out.print(s1.charAt(5)); // output: e
System.out.print(s1.charAt(12)); // output: k
Some Conversion methods

- `toLowerCase()` – returns all lower case version of string
- `toUpperCase()` – returns all upper case version of string
- `trim()` – returns a string that eliminates leading and trailing blank characters from original
- `replace()` – returns a string with an old character replaced with a new one. Old character and new character passed as parameters
Examples

String s1 = "Zebra"

String s2 = s1.toLowerCase(); // s2 is "zebra"
String s3 = s1.toUpperCase(); // s3 is "ZEBRA"

String s4 = " Apple ";

String s5 = s4.trim(); // s5 is "Apple"
String s6 = s5.replace('e', 'y'); // s6 is "Apply"
valueOf() method

- valueOf() – there are several of these methods.
- They are **static** methods, and are used for converting other values to String objects

```java
int x = 12345;

String s7 = String.valueOf(4.56); // s7 is "4.56"
String s8 = String.valueOf(16); // s8 is "16"
String s9 = String.valueOf(x); // s9 is "12345"
```
The StringBuilder Class

- The StringBuilder class is a part of the java.lang package.
- A StringBuilder object is mutable (i.e. it can be changed).
- Three of the four constructors shown here. Here are sample creations:
  - creates an empty string builder with initial capacity of 16 characters
    ```java
    StringBuilder buf1 = new StringBuilder();
    ```
  - creates empty string builder with initial capacity given in parameter
    ```java
    StringBuilder buf2 = new StringBuilder(50);
    ```
  - creates string builder filled with argument – initial capacity is length of given string plus 16
    ```java
    StringBuilder buf3 = new StringBuilder("Hello");
    ```
The append() method

- append() – adds data to string in the builder object, at the end. Several versions for different parameter types (see API for full set)

```java
StringBuilder buf1 = new StringBuilder();

buf1.append("Hello");
buf1.append(’,’);
buf1.append(" world!");
    // buf1 is now "Hello, world!"

buf1.append(’ ’);
buf1.append(123.45);
    // buf1 is now "Hello, world! 123.45"
```
The insert() method

- *insert()* – insert data at a certain starting index. Like append, multiple versions for different types of data (see API for full set)

```java
StringBuilder buf2 = new StringBuilder();

buf2.append("Welcome home");
    // buf2 now "Welcome home"

buf2.insert(8,"to my humble ");
    // buf2 = "Welcome to my humble home"
```
More StringBuilder methods

▶ delete() – delete data from a string builder object

```java
text here
```

▶ deleteCharAt() – delete a character at specified index

```java
text here
```

```java
StringBuilder buf3 = new StringBuilder("abcdefghijklm");

buf3.delete(4,9);
// deletes indices 4-8. buf3 is now "abcdjklm"

StringBuilder buf4 = new StringBuilder("abcdefg");

buf4.deleteCharAt(3); // buf4 is now "abcefg"
buf4.deleteCharAt(1); // buf4 is now "acefg"
```
More StringBuilder methods

- reverse() – reverses the contents of the string builder
- setCharAt() – sets a character at specified index (similar to deleteCharAt())
- capacity() – returns current capacity of builder
- length() – returns length of current string in builder (less than or equal to capacity)
- setLength() – sets the exact length of the string in the builder to new value (parameter).
  - This is the actual string, not the capacity.
  - If the new length is smaller than previous length, characters are truncated from the string.
  - If new length bigger, null characters are appended.
- charAt() – returns character at a specified index (parameter)