AP CSA Lecture 7

2022-11-21

String Class

String Objects

An object of type `String` is a sequence of characters. All *string literals*, such as `"yikes!"`, are implemented as instances of this class. A string literal consists of zero or more characters, including escape sequences, surrounded by double quotes. (The quotes are not part of the `String` object.) Thus, each of the following is a valid string literal:

```
""                  //empty string
"2468"
"I must\n go home"
```

`String` objects are *immutable*, which means that there are no methods to change them after they've been constructed. You can, however, always create a new `String` that is a mutated form of an existing `String`.

Create String objec

A `String` object is unusual in that it can be initialized like a primitive type:

```
String s = "abc";
```

This is equivalent to

```
String s = new String("abc");
```

in the sense that in both cases `s` is a reference to a `String` object with contents `"abc"` (see Box on p. 179).

It is possible to reassign a `String` reference:

```
String s = "John";
s = "Harry";
```

This is equivalent to

```
String s = new String("John");
s = new String("Harry");
```

Notice that this is consistent with the immutable feature of `String` objects. `"John"` has not been changed; he has merely been discarded! The fickle reference `s` now refers to a new `String`, `"Harry"`. It is also OK to reassign `s` as follows:

```
s = s + " Windsor";
```

`s` now refers to the object `"Harry Windsor"`.

Here are other ways to initialize `String` objects:

```
String s1 = null;           //s1 is a null reference
String s2 = new String();   //s2 is an empty character sequence

String state = "Alaska";
String dessert = "baked " + state;  //dessert has value "baked Alaska"
```

Check your understanding

How to create a String s: “bison”

How to create an empty String

How to add two String

What is the output of the following code:

int x = 2; String y = “2”;

print(x+y)

Compare two strings

1. equals
2. compareTo

There are two ways to compare `String` objects:

1. Use the `equals` method that is inherited from the `Object` class and overridden to do the correct thing:

   ```
   if (string1.equals(string2)) ...
   ```

   This returns `true` if `string1` and `string2` are identical strings, `false` otherwise.
2. Use the `compareTo` method. The `String` class has a `compareTo` method:

   ```
   int compareTo(String otherString)
   ```

   It compares strings in dictionary (lexicographical) order:

   - If `string1.compareTo(string2) < 0`, then `string1` precedes `string2` in the dictionary.
   - If `string1.compareTo(string2) > 0`, then `string1` follows `string2` in the dictionary.
   - If `string1.compareTo(string2) == 0`, then `string1` and `string2` are identical. (This test is an alternative to `string1.equals(string2)`.)

Be aware that Java is case-sensitive. Thus, if `s1` is `"cat"` and `s2` is `"Cat"`, `s1.equals(s2)` will return `false`.

Why we don't use "=="? Because "==" only compare reference

More about compareTo

Characters are compared according to their position in the ASCII chart. All you need to know is that all digits precede all capital letters, which precede all lowercase letters. Thus "5" comes before "R", which comes before "a". Two strings are compared as follows: Start at the left end of each string and do a character-by-character comparison until you reach the first character in which the strings differ, the *k*th character, say. If the *k*th character of s1 comes before the *k*th character of s2, then s1 will come before s2, and vice versa. If the strings have identical characters, except that s1 terminates before s2, then s1 comes before s2. Here are some examples:

```
String s1 = "HOT", s2 = "HOTEL", s3 = "dog";
if (s1.compareTo(s2) < 0))     //true, s1 terminates first
    ...
if (s1.compareTo(s3) > 0))     //false, "H" comes before "d"
```

Pop quiz:

String s1 = “abc”; String s2 = “abd”;

What is the output of the following code:

```
print(s1.compareTo(s2))


print(“Bc”.compareTo(“aa”))

print(“BD”.compareTo(“B”))

print(“1B”.compareTo(“Aa”))
```

Pop quiz

String s1 = “abc”; String s2 = “abc”;

What is the output of the following code:

```
print(s1 == s2)

print(s1.equals(s2))

print(s1.compareTo(s2))
```

Pop Quiz:

String s1 = “abc”; String s2 = “abc”;

s2 = s1

What is the output of the following code:

print(s1 == s2)

print(s1.equals(s2))

print(s1.compareTo(s2))

Pop quiz:

10. Consider these declarations:

```
String s1 = "crab";
String s2 = new String("crab");
String s3 = s1;
```

Which expression involving these strings evaluates to `true`?

I `s1 == s2`

II `s1.equals(s2)`

III `s3.equals(s2)`

(A) I only
(B) II only
(C) II and III only
(D) I and II only
(E) I, II, and III

Pop quiz:

11. Suppose that `strA = "TOMATO"`, `strB = "tomato"`, and `strC = "tom"`. Given that `"A"` comes before `"a"` in dictionary order, which is true?

    (A) `strA.compareTo(strB) < 0 && strB.compareTo(strC) < 0`

    (B) `strB.compareTo(strA) < 0 || strC.compareTo(strA) < 0`

    (C) `strC.compareTo(strA) < 0 && strA.compareTo(strB) < 0`

    (D) `!(strA.equals(strB)) && strC.compareTo(strB) < 0`

    (E) `!(strA.equals(strB)) && strC.compareTo(strA) < 0`

Important String method: length()

```
int length()
```

Returns the length of this string.

```
String s = "funnyfarm";

int y = s.length();          //y has value 9
```

indexOf(String str)

```
int indexOf(String str)
```

Returns the index of the first occurrence of `str` within this string. If `str` is not a substring of this string, `-1` is returned. The method throws a `NullPointerException` if `str` is null.

```
String s = "funnyfarm";
int x = s.indexOf("farm");  //x has value 5
x = s.indexOf("farmer");    //x has value -1
int y = s.length();         //y has value 9
```

substring

```
String substring(int startIndex)
```

Returns a new string that is a substring of this string. The substring starts with the character at `startIndex` and extends to the end of the string. The first character is at index zero. The method throws an `IndexOutOfBoundsException` if `startIndex` is negative or larger than the length of the string. Note that if you're using Java 7 or above, you will see the error `StringIndexOutOfBoundsException`. However, the AP Java subset lists only `IndexOutOfBoundsException`, which is what they will use on the AP exam.

```
String substring(int startIndex, int endIndex)
```

Returns a new string that is a substring of this string. The substring starts at index `startIndex` and extends to the character at `endIndex-1`. (Think of it this way: `startIndex` is the first character that you want; `endIndex` is the first character that you *don't* want.) The method throws a `StringIndexOutOfBoundsException` if `startIndex` is negative, or `endIndex` is larger than the length of the string, or `startIndex` is larger than `endIndex`.

```
"unhappy".substring(2)       //returns "happy"
"cold".substring(4)          //returns "" (empty string)
"cold".substring(5)          //StringIndexOutOfBoundsExceptio
"strawberry".substring(5,7)  //returns "be"
"crayfish".substring(4,8)    //returns "fish"
"crayfish".substring(4,9)    //StringIndexOutOfBoundsExceptio
"crayfish".substring(5,4)    //StringIndexOutOfBoundsExceptio
```

In the first case, the start index should not **exceed** the length. In the second case, the end index should not **exceed** the length.

Pop quiz:

12. This question refers to the following declaration:

```
String line = "Some more silly stuff on strings!";
//the words are separated by a single space
```

What string will str refer to after execution of the following?

```
int x = line.indexOf("m");
String str = line.substring(10, 15) + line.substring(25, 25 + x);
```

(A) "sillyst"
(B) "sillystr"
(C) "silly st"
(D) "silly str"
(E) "sillystrin"

Pop quiz:

13. A program has a `String` variable `fullName` that stores a first name, followed by a space, followed by a last name. There are no spaces in either the first or last names. Here are some examples of `fullName` values: `"Anthony Coppola"`, `"Jimmy Carroll"`, and `"Tom DeWire"`. Consider this code segment that extracts the last name from a `fullName` variable, and stores it in `lastName` with no surrounding blanks:

```
int k = fullName.indexOf(" ");     //find index of blank
String lastName = /* expression */
```

Which is a correct replacement for /* *expression* */?

I `fullName.substring(k);`

II `fullName.substring(k + 1);`

III `fullName.substring(k + 1, fullName.length());`

(A) I only
(B) II only
(C) III only
(D) II and III only
(E) I and III only

Pop quiz:

16. Consider this method:

```
public static String doSomething(String s)
{
    final String BLANK = " ";   //BLANK contains a single space
    String str = "";            //empty string
    String temp;
    for (int i = 0; i < s.length(); i++)
    {
        temp = s.substring(i, i + 1);
        if (!(temp.equals(BLANK)))
            str += temp;
    }
    return str;
}
```

Which of the following is the most precise description of what `doSomething` does?

(A) It returns `s` unchanged.
(B) It returns `s` with all its blanks removed.
(C) It returns a `String` that is equivalent to `s` with all its blanks removed.
(D) It returns a `String` that is an exact copy of `s`.
(E) It returns a `String` that contains `s.length()` blanks.