Fundamentals of Java Programs

(Part 3-Reference datatype -Strings)

Reference

- Readings
  - Chapter 14: Strings, Characters and Regular Expressions

Introduction to Object

- In Java and other object-oriented languages, an object is a collection of data that provides a set of methods. For example, Scanner, is an object that provides methods for parsing input. System.out and System.in are also objects.

- Strings are objects, too. They contain characters and provide methods for manipulating character data.

Primitive vs Reference Data Type

- Java's types are divided into primitive types and reference types.
  - Primitive types: `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` and `double`.
- All nonprimitive types are *reference types*.
- A primitive-type variable can hold exactly *one* value of its declared type at a time.
- Programs use variables of reference types (normally called references) to store the *addresses* of objects in the computer's memory.
  - Such a variable is said to refer to an object in the program.

Class String

- Class `String` is used to represent strings in Java.
- Create an empty string represented as "" and has a length of 0.
  - `String s1 = new String();`
- Constructor that takes a `String` object copies the argument into the new `String`.
  - `String str = "Hello";`
  - `String s = new String("Hello");`
  - `String s2 = new String(s2);`

```
1  // Fig. 14.1: StringConstructors.java
2  // String class constructors.
3
4  public class StringConstructors
5  {
6     public static void main(String[] args)
7     {
8        char[] charArray = {'b', 'i', 'r', 't', 'h', ' ', 'd', 'a', 'y'};
9        String s = new String("hello");
10
11       // use String constructors
12       String s1 = new String();
13       String s2 = new String(s);
14       String s3 = new String(charArray);
15       String s4 = new String(charArray, 6, 3);
16
17       System.out.printf(
18          "s1 = %s%ns2 = %s%ns3 = %s%ns4 = %s%n", s1, s2, s3, s4);
19    }
20 } // end class StringConstructors
```

Fig. 14.1 | `String` class constructors. (Part 1 of 2.)

```
s1 =
s2 = hello
s3 = birth day
s4 = day
```

Fig. 14.1 | `String` class constructors. (Part 2 of 2.)

Characters

- Strings provide a method named **charAt**, which extracts a character. It returns a char, a primitive type that stores an individual character.

```
String fruit = "banana";
char letter = fruit.charAt(0);
```

- The argument 0 means that we want the letter at position 0. Like array indexes, string indexes start at 0, so the character assigned to letter is b.
- String method **length** determines the number of characters in a string.

```
String fruit = "banana";
System.out.print(fruit.length()); //output 6
System.out.print( fruit.charAt(fruit.length())); //Error!
System.out.print( fruit.charAt(fruit.length()-1)); //Correct!
```

Strings are Immutable

- Strings provide methods, ***toUpperCase*** and ***toLowerCase***, that convert from uppercase to lowercase and back.
- These methods are often a source of confusion, because it sounds like they modify strings. But neither these methods nor any others can change a string, because strings are **immutable**.

```
String name = "Hello Java";
String upperName = name.toUpperCase();
System.out.print(name +"\n"+ upperName);
```

Output:

```
Hello Java
HELLO JAVA
```

Substrings Method

- The **substring** method returns a new string that copies letters from an existing string, starting at the given index.
  - fruit.substring(0) returns "banana"
  - fruit.substring(2) returns "nana"
  - fruit.substring(6) returns "“
- There are other versions of substring that have different parameters. If it’s invoked with two arguments, they are treated as a start and end index:
  - fruit.substring(0, 3) returns "ban"
  - fruit.substring(2, 5) returns "nan"
  - fruit.substring(6, 6) returns ""

The indexOf Method

- The **indexOf** method searches for a character in a string and it returns the index of the first appearance.

```
String fruit = "banana";
int index = fruit.indexOf('a’); //index=1
```

- To find subsequent appearances, you can use another version of **indexOf**, which takes a second argument that indicates where in the string to start looking.

```
int index = fruit.indexOf('a', 2); //index=3
```

- If the character does not appear in the string, **indexOf** returns -1. Since indexes cannot be negative, this value indicates the character was not found.

The indexOf Method

- You can also use **indexOf** to search for a substring, not just a single character.

```
int index = fruit.indexOf("nan"); //index=2
```

- Method **lastIndexOf** locates the last occurrence of a character in a String. The method searches from the end of the String toward the beginning. If it finds the character, it returns the character's index in the String—otherwise, it returns –1.
- A second version of **lastIndexOf** takes two integer arguments—the integer representation of the character and the index from which to begin searching backward.

```
String fruit = "banana";
int index = fruit.lastIndexOf('a'); //index=5
int index = fruit.lastIndexOf('a',4); //index=3
```

String Comparison

- **equals** method used to compare strings.
- The equals method returns true if the strings contain the same characters; otherwise it returns false.
- String method **equalsIgnoreCase** ignores whether the letters in each String are uppercase or lowercase when performing the comparison.

```
String name1 = "Java";
String name2 = "JAVA";

boolean answer= name1.equals(name2);
System.out.println("Answer="+answer); //output false

answer= name1.equalsIgnoreCase(name2);
System.out.println("Answer="+answer); //output true
```

String Comparison

- We can use **compareTo** to see which comes first in alphabetical order. The return value from **compareTo** is the difference between the first characters in the strings that differ.
  - If the strings are equal, their difference is zero.
  - If the first string (the one on which the method is invoked) comes first in the alphabet, the difference is negative. Otherwise, the difference is positive.

```
String name1 = "Alan Turing";
String name2 = "Ada Lovelace";
int answer= name1.compareTo(name2);
System.out.println("Answer="+answer); //output positive 8
```

  - In the preceding code, compareTo returns positive 8, because the second letter of "Ada" comes before the second letter of "Alan" by 8 letters.
  - Both **equals** and **compareTo** are case-sensitive. The uppercase letters come before the lowercase letters, so "Ada" comes before "ada".

String Checking

- String methods **startsWith** and **endsWith** determine whether strings start with or end with a particular set of characters.

```
String name1 = "started";
String name2 = "start";
String name4 = "end";

System.out.println(name1.startsWith('s')); //output true
System.out.println(name1.startsWith("st")); //output true

System.out.println(name2.startsWith('s')); //output true
System.out.println(name2.startsWith("st")); //output true

System.out.println(name1.endsWith('d')); //output true
System.out.println(name1.endsWith("ed")); //output true

System.out.println(name2.endsWith('d')); //output false
System.out.println(name3.endsWith('d')); //output true
```