Activity 1.1.4 If It's Raining…

Otakar Andrysek

Subtarget 1.3: Understand what it means for a variable to reference an object, how the equals() method differs from the == conditional, and how to use common String methods such as length, substring, indexOf and compareTo.

I will look at your responses to 7, 8, 10, 12, 17 and 32. I will post an answer key for you to check your other responses on your own.

Introduction

So far you have learned about classes and different types of variables in Java. In this activity, you will continue learning how data is represented in Java by going deeper into strings and the String class. Strings are a type of object that group individual characters together. You will use strings any time you want to represent data, like names of people, places, or things. You will also use strings when you want to display text or error messages in an Android app.

In this activity, you will also create **conditional** statements. If you've ever been told, "You can only have dessert if you eat your dinner," then you've already got some real-life experience with conditionals. In this activity you will create the logic behind an app that can advise people on a course of action based on the weather in your town or city (If it's raining, take an umbrella).

Materials

- Computer with BlueJ and Android™ Studio
- ~~Android™ tablet and USB cable, or a device emulator~~

Activity

Part I: Creating Strings

You already created one String literal in *Activity 1.1.1 Introduction to Android Development*, when you typed "Hello World!". You learned that the System.out.println method takes a String as its argument and displays that output to a console. In this activity you will continue learning about the methods associated with the String class.

1. Review **What is a String?** and answer the following questions:

    a. How can you tell that Strings are objects and not a primitive type?

    Because the S is capitalized, meaning that a String is an object.

    b. What does **null** mean?

    Absence or 'lack of' information.

    c. In what two ways could you create a String that has the value "This is a test message"?

    String x11 = "Hello World!";

    String x2;
    x2 = "Hi World!"

    d. What is a **superclass**?

    A class that contains classes within in.

    e. What is the superclass of String?

    The Object Class

    f. What method can you call to determine what class an object belongs to?

    Class parentClass = currClass.getSuperclass();

When you declare and initialize a string, such as String s = "This is a test.", the data for the String object is actually a sequence of characters, stored one after the other in memory:

T	h	i	s		i	s		a		t	e	s	t	.

You can “index” into this sequence of characters numerically, starting at index 0. The character T is at index 0, character c is at index 1, character i is at index 2, and so on until you get to the last character . at index 14. The String class provides methods that manipulate the characters using these index values.

2. Review String Methods and answer the review questions to make sure you understand the String methods described. You’ll need a working understanding of them for the next steps.

3. Open BlueJ and create a new project called “Weather”.

   Done.

4. Create a new class called StringTester, deleting the auto-generated code and replacing it with a main method as you have done previously.

   Done.

5. Using **String Methods** as a reference, create a new String called weatherCondition. Give it a value of one of the descriptions from the “Conditions Codes” table in **Yahoo! Weather condition codes**.

   Done.

6. What statement would you write to print to the console the number of characters in weatherCondition?

   System.out.println(weatherCondition.length());

7. Write an output statement that uses the substring method in such a way that the first word of condition codes 5, 6, 7, 14, 18, 22, 24, 31, 32, 35, 41, and 43 will print correctly. (Hint: Why would it be the same command for each of these codes?)

   Because for all of these condition codes the first word is five characters long.

8. If called when weatherCondition stores the value of condition code 39, what would the following statement print?
   System.out.println(weatherCondition.indexOf("thunder"));

   10 (This is the segment at the beginning of string “thunder”)

Part II: Weather Advice

A **Boolean expression** is an expression that evaluates to true or false, such as “It is raining outside”, or in a more Java-like statement, “weather is raining”. In this part of the activity, you will use Boolean expressions and string methods to recommend a course of action based on the weather. For example, you can determine, “If it is raining, take your umbrella with you!”

9. **Learn to use the String equals(…) method** and be sure to answer the review question.

   Done.

10. Given the code below, explain in your own words the difference between the result1 and result2.

```
1  String weatherCondition1 = "mixed rain and snow";
2  String weatherCondition2 = "mixed rain and snow";
3
4  boolean result1 = (weatherCondition1 == weatherCondition2);
5  boolean result2 = (weatherCondition1.equals(weatherCondition2));
```

    Result one is comparing the weatherCondition objects, resulting in false as the objects are not located in the same place in memory. Result two is comparing the contents of both weatherCondition Strings, returning true.

11. Later in this activity, you will retrieve the current weather condition from **Yahoo! Weather** and provide a user some advice depending on what that condition is. Say you have a String called currentCondition. How would you check to see if that String contained the exact value "heavy snow”?

    boolean isSnowing = (currentCondition.equals(“heavy snow”));

12. True or False? compareTo returns –1 when the value of the current String is less than the other String.

    this.compareTo(that) When (this < that) a negative value is returned.
    True.

13. Review the material on **String concatenation**. You will use **concatenation** to create your advice statement.

    Fun!

14. In your BlueJ project, create a new class called WeatherConditionals.

Yes master.

15. Replace the default constructor and method with the following:

```
1  public class WeatherConditionals
2  {
3      public static String getWeatherAdvice(int temperature, String
   description){
4          return ;
5      }
6  }
```

Complete.

Between the keyword return and the semicolon on line 4 (within the body of the getWeatherAdvice method), there should be an expression that evaluates to a String. To construct the String, you will use the concatenation operator. For example, if the value of temperature is 32 and the description is "heavy snow", the return value would be "32 degrees and heavy snow." (Don't forget the period.)

All done.

16. In StringTester, within the main method, add the following just before the closed curly brace:

```
1  Sysetm.out.println(WeatherConditionals.getWeatherAdvice(32, "heavy
   snow"));
```

Inserted.

17. Run the main method of the StringTester class to verify that your work in Step 14 was correct.
    Paste a snip of your getWeatherAdvice method and the output from running the main.

    You can also check my GitHub!
    https://github.com/otakar-sst/CSA/tree/master/Java%20Programs/Lesson%201/1.1.4/Weather

```
// Begin the actual class
public class WeatherConditionals
{
    // Create a method to give weather advice based on temperature and weather
    public static String getWeatherAdvice(int temperature, String description){
        // Return a combined string in format: "32 degrees and heavy snow."
        return temperature + " degrees and " + description + ".";
    }
}
```

```
scattered thunderstorms
23
scatt
10
32 degrees and heavy snow.
```

Part III: Conditional Weather

In this part of the activity, you will review Conditionals and learn about complex conditionals, which can be helpful in situations where you want to respond to the status of a combination of conditions.

18. Review information on **Conditionals**, answering the review questions on the website to verify that you have learned what you need.

    Done.

19. Within the getWeatherAdvice method of the WeatherConditionals class, remove your return statement and create a variable of primitive type boolean with the identifier windy and a value false.

    Done.

20. Add a conditional statement to your program to determine if the string description contains “windy” and set windy appropriately.

    Consider it done.

21. Use windy and temperature in another conditional statement to determine if it is not windy and also warm enough (more than 30 degrees) to go outside. Test your program for the following results:

temperature	description	result
34	sunny	It's safe to go outside, 34 degrees and sunny.
32	windy	Too windy or cold! Enjoy watching the weather through the window.
33	snow	It's safe to go outside, 33 degrees and snow.
30	snow	Too windy or cold! Enjoy watching the weather through the window.
30	windy	Too windy or cold! Enjoy watching the weather through the window.

    I hope I did this the way I was supposed to…

22. Review **Complex Conditionals**, answering the review questions to verify that you have learned what you need.

    Easy peasy

23. Add a conditional statement so that your getWeatherAdvice method can determine a weather condition where the description contains “snow” and the temperature is over 100 degrees. Return a message expressing disbelief at this combination. Test your program.

    Tested.

24. Assume you can store more than one weather condition at a time, and that you don’t like to go out if it is both freezing and cloudy. One or the other is fine, but not both. In computer science you can evaluate this kind of condition using **short circuit evaluation**. Read about **short circuit evaluation** and answer the review question.

    Review complete.

25. Assume the boolean variables freezing, cloudy, fair, and sunny. Determine the values that would cause a short circuit evaluation in the following statements.
    a. if (freezing && cloudy)

       freezing = false

    b. if (sunny || fair)

       sunny = true

    c. if (!sunny && !fair)

       sunny = true

26. Sometimes rewriting a conditional expression can make it easier for humans to read or understand. **Learn DeMorgan’s Law** and answer the review questions.

    Understood (kind of).

27. Given the boolean variables sunny, clear, raining, and snowing, rewrite the following conditional expressions using **DeMorgan’s Law**.
    a. if (!sunny || !clear)

       if (!(sunny && clear))

    b. if (!(!raining && !snowing))

       if (!(raining || snowing))

28. Similar to DeMorgan's Law, you can rewrite relational operators when they are used with the not operator !. For example, "not less-than" is the same as "greater-than-or-equal-to". Rewrite the following conditional expressions without using !.
    a. if (!(temperature > 75))

       if (temperature <=75)

    b. if (!(temperature <= 100))

       if (temperature > 100)

    c. if (!(temperature == 32))

       if (temperature != 32)

       OR

       if (temperature < 32 || temperature >32)

Part IV: Planning for a Weather App

Now that you have the necessary knowledge of conditionals, program your app to make recommendations based on several weather indications.

29. A client wants an app that provides guidance as they prepare to go for a hike in the morning. You have access to the following information:
    - temperature as an int
    - windchill as an int
    - humidity as an int
    - description as a String

    The temperature and windchill units are Fahrenheit, and humidity represents a percentage. The description will be one of the **Yahoo! Weather conditions** in the table referenced in Step 5. Plan out how you would advise this client based on these inputs.

    Planned out on paper.

30. As directed by your teacher, work with a partner to refine your plan for advising the hiker. Determine favorable (or unfavorable) hiking conditions, such as rain, heat, cold, and the other conditions listed.
    Here are the seven possible bits of advice; your app should determine the appropriate line of advice to give based on the info (You can re-word them if you like):

"Weather is good. Looks like another great day for a hike!"
"It's too hot or hot and humid for a hike; wait for a cooler day."
"Dangerous weather possible; best to stay sheltered."
"It's too cold outside; wait for it to warmup."
"Rain or showers forecast; not a good hiking day."
"Might be too smoky for a hike; best to check first"
"There may be drizzle; maybe take a rain coat."

    Good ideas, will note.

31. When you are ready, implement a new method within WeatherConditionals using the signature shown below and filling in the body of the method (line 3) with conditionals that you designed in the previous two steps.

```
1     public static String getHikingAdvice(int temperature, int windchill,
int humidity, String description){
2
3
      }
4
```

32. Call your method from StringTester, passing in various values to make sure that you have tested all of your **boundary conditions**. Testing boundary conditions means that you should test all of the conditions in your if statements, confirm that the correct code is executed, and that *all* statements can be reached or executed.
    Paste snips of the following:
    a) your getHikingAdvice method,

    No way! It's about 550 lines. See my GitHub.
    https://github.com/otakar-sst/CSA/blob/master/Java%20Programs/Lesson%201/1.1.4/Weather%20App/WeatherConditionals.java

    b) a snip of all of your consecutive calls of this method in the main that test the boundary conditions,

    Essentially every possible outcome is a boundary condition for the way I did it. I'll just test a few.

```
    // Set the weather conditions
    WeatherConditionals.getHikingAdvice(49,43,65,"partly cloudy");
    WeatherConditionals.getHikingAdvice(89,43,65,"thunderstorm");
    WeatherConditionals.getHikingAdvice(89,43,85,"partly cloudy");
    WeatherConditionals.getHikingAdvice(41,23,55,"sunny");
    WeatherConditionals.getHikingAdvice(57,54,92,"sunny");
    WeatherConditionals.getHikingAdvice(49,43,65,"rain");
}
```

c) a snip of your output after running the main

```
Hello and welcome to my hiking weather system.
I will give you a reccomendation based on the current weather
Currently it is partly cloudy and 49 degrees
The windchill is 43 degrees and the humidity is 49

PERFECT! GO, GO NOW


Hello and welcome to my hiking weather system.
I will give you a reccomendation based on the current weather
Currently it is thunderstorm and 89 degrees
The windchill is 43 degrees and the humidity is 89

It's thundering out there, stay at home


Hello and welcome to my hiking weather system.
I will give you a reccomendation based on the current weather
Currently it is partly cloudy and 89 degrees
The windchill is 43 degrees and the humidity is 89

It's nasty out there, stay at home.


Hello and welcome to my hiking weather system.
I will give you a reccomendation based on the current weather
Currently it is sunny and 41 degrees
The windchill is 23 degrees and the humidity is 41

A little windy, but great otherwise!


Hello and welcome to my hiking weather system.
I will give you a reccomendation based on the current weather
Currently it is sunny and 57 degrees
The windchill is 54 degrees and the humidity is 57

A little humid, but still a perfect hiking day.


Hello and welcome to my hiking weather system.
I will give you a reccomendation based on the current weather
Currently it is rain and 49 degrees
The windchill is 43 degrees and the humidity is 49

It's a GREAT walk... in the rain
```

Conclusion

1. Create boundary conditions using an if statement to ensure that a String str is neither empty nor null, and that it has no more than 80 characters. Note: There are at least two ways to write this—can you come up with two?

   Option 1

```
if (str != null && !str.isEmpty())
{
        // Do something
}
```

   Option 2

```
if (string.equals(null) || string.equals("")
{
        // Do something
}
```

2. Evaluate the opposite of one of your statements above by putting a not (!) in front of the statement and applying DeMorgan's Law to simplify the statement.

   The opposite of Option 1:

```
if (!(str = null || str.isEmpty()))
{
        // Do something
}
```