Activity 1.1.4 If It’s Raining…

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?**](http://interactivepython.org/runestone/static/JavaReview/Strings/sbasics.html) and answer the following questions:
   1. How can you tell that Strings are objects and not a primitive type?
      1. It starts with a capital letter.
   2. What does null mean?
      1. It’s a value that means that there is no object associated with this variable – it basically means that there is *literally nothing*.
   3. In what two ways could you create a String that has the value "This is a test message"?
      1. String m1 = “This is a test message”;
      2. String m2 = new String(“This is a test message”);
   4. What is a superclass?
      1. A class that is the parent class of the class. The class inherits behaviors from the superclass.
   5. What is the superclass of String?
      1. java.lang.Object
   6. What method can you call to determine what class an object belongs to?
      1. instance.getClass();

When you declare and initializea 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.

1. Review [**String Methods**](http://interactivepython.org/runestone/static/JavaReview/Strings/sMethods.html) 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.
2. Open BlueJ and create a new project called “Weather”.
3. Create a new class called StringTester, deleting the auto-generated code and replacing it with a main method as you have done previously.
4. Using [**String Methods**](http://interactivepython.org/runestone/static/JavaReview/Strings/sMethods.html) 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**](https://developer.yahoo.com/weather/documentation.html#codes).
5. What statement would you write to print to the console the number of characters in weatherCondition?
   1. System.out.println(weatherCondition);
6. 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.
7. If called when weatherCondition stores the value of condition code 39, what would the following statement print? System.out.println(weatherCondition.indexOf("thunder"));

10

**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!”

1. [**Learn to use the String equals(…) method**](http://interactivepython.org/runestone/static/JavaReview/Strings/sEquality.html) and be sure to answer the review question.
2. Given the code below, explain in your own words the difference between the result1 and result2.

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

1. Later in this activity, you will retrieve the current weather condition from [**Yahoo! Weather**](https://developer.yahoo.com/weather/documentation.html#codes) 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"?
   1. currentCondition.equals(“heavy snow”);
2. True or False? compareTo returns –1 when the value of the current String is less than the other String.
3. Review the material on [**String concatenation**](http://interactivepython.org/runestone/static/JavaReview/Strings/sOperators.html). You will use concatenation to create your advice statement.
4. In your BlueJ project, create a new class called WeatherConditionals.
5. Replace the default constructor and method with the following:

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

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.)

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

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

1. Run the main method of the StringTester class to verify that your work in Step 14 was correct.

**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.

1. Review information on [**Conditionals**](http://interactivepython.org/runestone/static/JavaReview/Conditionals/cbasics.html), answering the review questions on the website to verify that you have learned what you need.
2. 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.
3. Add a conditional statement to your program to determine if the string description contains “windy” and set windy appropriately.
4. 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. |

1. Review [**Complex Conditionals**](http://interactivepython.org/runestone/static/JavaReview/Conditionals/cComplex.html), answering the review questions to verify that you have learned what you need.
2. 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.
3. 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**](http://interactivepython.org/runestone/static/JavaReview/Conditionals/cShortCircuit.html) and answer the review question.
4. Assume the boolean variables freezing, cloudy, fair, and sunny. Determine the values that would cause a short circuit evaluation in the following statements.
   1. if (freezing && cloudy)
      1. if freezing is false, cloudy will not be evaluated
   2. if (sunny || fair)
      1. if sunny is true, fair will not be evaluated
   3. if (!sunny && !fair)
      1. if sunny is true, fair will not be evaluated
5. Sometimes rewriting a conditional expression can make it easier for humans to read or understand. [**Learn DeMorgan’s Law**](http://interactivepython.org/runestone/static/JavaReview/Conditionals/cDeMorgans.html) and answer the review questions.
6. Given the boolean variables sunny, clear, raining, and snowing, rewrite the following conditional expressions using **DeMorgan’s Law**.
   1. if (!sunny || !clear)
      1. !(sunny && clear)
   2. if (!(!raining && !snowing))
      1. raining || snowing
7. 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 !.
   1. if (!(temperature > 75))
      1. temperature <= 75
   2. if (!(temperature <= 100))
      1. temperature > 100
   3. if (!(temperature == 32))
      1. 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.

1. 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**](https://developer.yahoo.com/weather/documentation.html#codes) in the table referenced in Step 5. Plan out how you would advise this client based on these inputs.

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

1. 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.

**Part V: Android Weather Notifier**

In this part of the activity, you will transfer your getHikingAdvice method into an Android Studio project. It will post a notification to the device’s screen every day at 5:00 PM, when the hiker gets done with work, advising the hiker about the current conditions.

1. If you have not opened Android Studio before, refer to Part III in Activity 1.1.1 Introduction to Android Development to launch Android Studio for the first time.
2. Create a WeatherAdvisor folder in your AndroidProjects folder.
3. Get a copy of the 1.1.4WeatherAdvisorApp Android project from your teacher. Copy or extract the files to a WeatherAdvisor folder in your AndroidProjects folder.
4. In Android Studio, import the WeatherAdvisor project: Select File > New > Import Project…
5. A dialog appears showing your file structure. Navigate to your AndroidProjects folder and then navigate to the location where you copied or extracted the WeatherAdvisor files. In the WeatherAdvisor file structure, select a file named build.gradle. It will be in the WeatherAdvisor folder, not in a subfolder. Click OK.
6. Find the WeatherConditionals.java file and paste your code from the last part of this activity into that file.
7. Build and run this app; describe its behavior.

**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?
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.