

ACTIVITY 1.1.4

If It's Raining...

INTRODUCTION

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, such as 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 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. 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.)

conditional

A statement or expression that controls the path of execution, depending on whether the expression evaluates to true or false.

Materials

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

RESOURCES




Lesson 1.1 Reference Card for Basic Constructs

Resources available online

Procedure

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:
- How can you tell that `Strings` are objects and not a primitive type?
 - What does **null** mean?
 - In what two ways could you create a `String` that has the value `"This is a test message"`?
 - What is a **superclass**?
 - What is the superclass of `String`?
 - What method can you call to determine what class an object belongs to?

null

A keyword in Java that indicates that an object variable has not been assigned a value—in other words, an object reference has not been created.




superclass

The parent, more general class in a class hierarchy.

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 `h` 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”.
- 4 Create a new class called `StringTester`, deleting the auto-generated code and replacing it with a `main` method as you have done previously.
- 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**.
- 6 What statement would you write to print to the console the number of characters in `weatherCondition`?


- 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.
- 8 If called when `weatherCondition` stores the value of condition code 39, what would the following statement print? `System.out.println(weatherCondition.indexOf("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!”

Boolean expression

An expression that evaluates to either true or false, as in $(x > 0)$.

- 9  **Learn to use the `String equals(...)` method** and be sure to answer the review question.
- 10 Given the code below, explain in your own words the difference between the `result1` and `result2`.

```
String weatherCondition1 = new String("mixed rain and snow");
String weatherCondition2 = new String("mixed rain and snow");
boolean result1 = (weatherCondition1 == weatherCondition2);
boolean result2 = (weatherCondition1.equals(weatherCondition2));
```

- 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”?

Check your answer

```
System.out.println(currentCondition.equals("heavy snow"));
```

- 12 Answer the following True/False question.

`compareTo` returns `-1` when the value of the current `String` is less than the other `String`.

- a. True
- b. False

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

concatenation

Adding two `String` objects together using the `+` operator.

- 14 In your BlueJ project, create a new class called `WeatherConditionals`.
- 15 Replace the default constructor and method with the following:

```
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.)


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

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



- 17 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.


- 18 Review information on  **Conditionals**, answering the review questions on the website to verify that you have learned what you need.
- 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`.
- 20 Add a conditional statement to your program to determine if the string `description` contains “windy” and set `windy` appropriately.
- 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.

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

short circuit evaluation

A complex conditional expression where the subsequent condition(s) might not be executed.

- 25 Assume the boolean variables `freezing`, `cloudy`, `fair`, and `sunny`. Determine the values that would cause a short circuit evaluation in the following statements.
- `if (freezing && cloudy)`
 - `if (sunny || fair)`
 - `if (!sunny && !fair)`
- 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.
- 27 Given the boolean variables `sunny`, `clear`, `raining`, and `snowing`, rewrite the following conditional expressions using **DeMorgan's Law**.
- `if (!sunny || !clear)`
 - `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 `!`.
- `if (!(temperature > 75))`
 - `if (!(temperature <= 100))`
 - `if (!(temperature == 32))`


DeMorgan's Law

A method you can use to rewrite the negative version of complex conditionals, for example `!(cold && raining)`.

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.

- 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.
- 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,  
2:           int windchill, int humidity, String description){  
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.

boundary conditions

In Java, the `if` statements that limit or define the execution path of code.

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.

- 33 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.
- 34 Create a *WeatherAdvisor* folder in your *AndroidProjects* folder
- 35 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.
- 36 In Android Studio, import the *WeatherAdvisor* project: Select **File > New > Import Project...**
- 37 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**.
- 38 Find the *WeatherConditionals.java* file and paste your code from the last part of this activity into that file.
- 39 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.