**Why Are You Stringing Me Along?**

Numbers can be compared with conditional statements and relational operators, but what about strings of characters? For example, what is your prediction about the following code segment?

String courseName = "AP Computer Science A";  
if(courseName == "AP Computer Science A")  
System.out.println("You are in the right class.");  
else  
System.out.println("Better check your schedule.");

This is the same pattern used to compare numeric values, so this conditional statement is written to try and determine if the contents of the **courseName**object contains the String literal in quotes. However, as you will shortly see, appearances can be deceiving because Java does not handle primitive data types and **String** objects in the same way when it comes to issues of equality. With objects of any type, two kinds of equality comparisons are possible: identity and content.

* **Identity equality** refers to whether two objects are actually the same object.
* **Content equality** indicates whether two objects contain equal values.

This lesson deals primarily with content equality for which relational operators will not work. Instead, Java provides a host of **String** class methods to evaluate and manipulate the contents of **String** reference variables.

### Part 1

Regardless of the number of alternative outcomes involved, writing condition statements with primitive data types is straightforward. One of three structures will apply (i.e., **if**, **if-else**, or **if-else-if**) and relational operators (i.e., <, >, <=, >=, ==, !=) give you six different ways to compare the relative equality of primitive data types. However, content equality of **String** objects must be compared with methods of the **String** class, not with relational operators.

The simplest way to evaluate the equality of **String** objects is with the **equals()** method, which is summarized in the Java API entry shown below:

boolean       **equals(Object** anObject)  
                     Compares this string to the specified object

The **equals()** method takes one parameter, which will be a **String** object for now, and returns a **boolean** value that can only be **true** or **false**. Case matters when evaluating **Strings**, so there are additional methods that deal with uppercase and lowercase information, as well as some methods that simply ignore case altogether.

Strings are objects! Always use the String methods to compare their values. If relational operators are used, you may get some unexpected results. Try running the code below. You'll discover str1 and str2 both have a content value of "A", but comparison with the relational operator yields false and the comparison with the String method yields true.

String str = "APCS";  
String str1 = "" + str.charAt(0);  
String str2 = "" + str.charAt(0);  
System.out.println("The string values: " + str1 + " and " + str2);  
System.out.println(" str1 == str2 : " + (str1 == str2) );  
System.out.println(" str1.equals(str2) : " + str1.equals(str2) );

### Part 2

In the following segment of code, notice the inclusion of a pair of curly braces around the blocks of code in each section of the conditional statement.

…   
String category = "";  
double admissionFee = 6.00;  
…   
if(age <= 10)  
**{**   
    admissionFee = 4.50;  
    category = "Child";  
**}**  
else  
**{**  
    admissionFee = 6.00;  
    category = "Adult";  
**}**

admissionFee += admissionFee \* tax;**…**

Each pair of curly braces defines a block of code containing multiple lines. Notice that all lines **within a block** end with a semicolon. Errors will occur if blocks include multiple lines without the curly braces or if semicolons are not included at the end of each statement. Even when a block contains only one statement, many programmers include the curly braces to emphasize the logical structure of the condition, as indicated in the following code segment:

if(age <= 10)  
{   
    admissionFee = 4.50;  
}  
else  
{  
    admissionFee = 6.00;  
}

To build consistent programming skills, it is suggested that you always use a pair of curly braces in conditional blocks, no matter how many statements are included.

### Part 2

**String** class methods are frequently necessary to process alphanumeric data, and you will use them throughout the course. Some of the most useful **String** methods include the following:

equals()

equalsIgnoreCase()

toUpperCase()

toLowerCase()

compareTo()

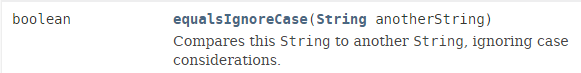
compareToIgnoreCase()

The descriptive names give a clue about the purpose of each method, although there may be some surprises. Be sure to examine the Detail and Summary sections of the **String** class in the Java API for further information about these methods.

## 03.06 Virtual Lecture NotesThe **String** class contains many methods that are directly and indirectly related to determining whether two **Strings** contain equal content. Download the demo files for each of the following methods and examine the source code. Experiment with each program until you understand how the methods work.

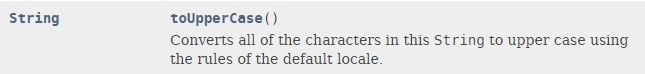
String methods: **equals()** **and equalsIgnoreCase()**

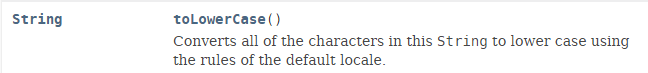
boolean. equals(Object anObject). Compares this string to the specified object.



Use the EqualsDemo.java file to test the methods. A **boolean** value of true is returned if the content value of the String object the method was invoked on matches the content value of the String object passed as a parameter. Otherwise false is returned. Notice, the **equals()**method is case sensitive.

String methods: **toUpperCase() and toLowerCase()**

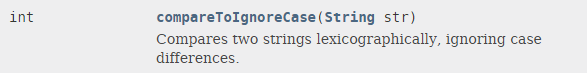




Use the ToCaseDemo.java file to discover how these methods work. The **toUpperCase()** and **toLowerCase()** methods are pretty straightforward. The String value returned will be the content value of the String object in which the method was invoked with all the characters as all uppercase or all lowercase.

String methods: **compareTo() and compareToIgnoreCase()**

int. compareTo(String anotherString). Compares two strings lexicographically.

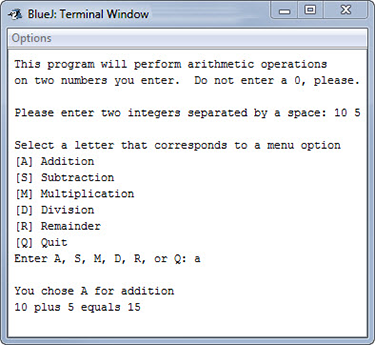


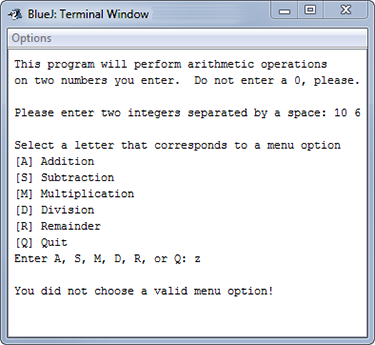
Time to run the CompareTo.java file. Be sure to use different input values and look close at the output. Do you notice any patterns? The **compareTo()** method will test the content value of the String the method is invoked on to the content value of the String passed as a parameter. If the first String alphabetically comes before the second String, a negative integer is returned. If the first String would be alphabetized after the second String, a positive integer is returned. If both Strings have the same content value, the integer 0 is returned.

### Part 3

Single characters are commonly used as input for text-based menus, as illustrated in the first screen shot image shown below. An alphanumeric character represents a specific menu option, and choosing the symbol causes the program to execute a specific block of code.

However, notice what character was entered for the Addition menu choice. Although the user did not follow directions and entered a lowercase letter, the program did not crash.





The second screen shot image shows a common user entry mistake: incorrect input. Once again, the program did not crash even though one of the menu options was not chosen. Instead, the program simply terminated in a user-friendly way with a notice to the user about the incorrect entry.

MathMenu

1. Create a new project called 03.06 Menu Selections in the Mod03 Lesson folder.
2. Download the [**MathMenu.java**](https://lti.flvsgl.com/flvs-cat-content/gpvnmt06qphi3ifocc58jt1jm1/flvs-cat-session/apcomputersciencea_v20/module03/lesson06/docs/04_06b/MathMenu.java) file to the newly-created folder.
3. Open the file, study the source code, and look carefully at the program comments.
4. After you understand the purpose and design of the program, complete the missing code so that all menu options work correctly.
5. Include an option to handle incorrect data entry.

When entry errors are made, the temptation among newbie programmers is to respond to the user with sarcastic comments. Although this can be entertaining for personal use, treat the user more respectfully in a program that goes out to the public!