We ask you to investigate some functions by writing calls on them. You can do this most easily in DrJava’s interactions pane. If you don’t have Dr.Java, then start a new project in Eclipse, add a class Check, and then replace its text by the following. Then, you can change the argument Character *toUpperCase*('a') of the println statement to print out what you want and run method main.

public class Check {

 /\*\* checking method calls. \*/

 public static void main(String[] args) {

 System.*out*.println(Character.*toUpperCase*('a'));

 }

}

**1. Class Math**. This part concerns using the API specs to learn about class Math. Use whatever method you want to bring up the API spec for class Math in your favorite browser. Simple ways are (1) use a link on the course website and (2) type the following into the search engine field of your browser: Java Math 8.

**1A.** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. To the left, write the name of the package that contains class Math.

**1B.** Read the first sentence in the description of class Math to see what the class contains. No need to read the rest of the description, for it contains details that you don’t need to know.

**1C.** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. To the left, write the names of the two static fields in class Math. Evaluate the expression Math.PI (in either DrJava or Eclipse) and write its value on the line.

**1D.** Scroll through the Method Summary to get an idea of a the functions in Math. They are all **static**, so you can call them, for example, Math.max(5, 3) — by preceding the function call with “Math.” (i.e. the name of the class followed by a period).

**1E**. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. On the line to the left, write the number of functions named Math.max. There are several; each with different parameter types. We say that the function with name max is *overloaded*. When you write a call on max, the types of the arguments are used to determine which function to call.

**1F.** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. To the left, write the name of the function that returns the “sign” of a value: -1.0 if the value is negative, 0.0 if 0, and 1.0 if positive. There are actually two such functions; one returns a double and the other a float. Note that 1.0f is a float value while 1.0 is a double value. Types are important in Java!

**2. Class Character**. Bring up the API spec for class Character in your favorite browser. Read the first two sentences of its description, which explain the two reasons for the class: Be able to treat a **char** value as an object (of class Character) and provide methods to deal with **char**s.

Many methods in class Character have a parameter named codePoint. Do not concern yourself with those methods. We don’t use them. Below, we explain why. Skip the explanation for now if you want and go right to point **2A** below.

The *Unicode Standard* defines how characters are represented. Google it to find out about it. The standard is set by an organization that is in charge, globally, of character representations. When Java was first defined, Unicode was a fixed-width 16-bit character (2-byte) encoding. It allowed for 65,536 characters. But more and more characters were added to the Unicode Standard, and soon more than 65,536 had to be represented! So the Unicode Standard was changed to use two 16-bit words for *some* characters, the so-called *supplementary* characters. Now, Unicode can allow up to 1,112,064 characters.

But Java type **char** was not changed. That would be too difficult. So, type **char** does not contain *all* characters, just as type **int** does not contain all integers.

Java *did* develop a way to handle all characters, and you can read about it here: [http://www.oracle.com/ us/technologies/java/supplementary-142654.html](http://www.oracle.com/). Basically, they use **ints** and not **chars**. It’s complicated. Methods in class Character that have a parameter named codePoint are designed to work with all characters, including the supplementary ones. No need for you to trouble yourself over this issue. It’s beyond the scope of our course.

**2A. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_.** On the line to the left, write the integer that represents the character ‘0’ and the integer that represents ‘1’. You can find them by casting ‘0’ and ‘1’ to type **int**.

**2B**. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. On the line to the left, write an expression that gives the numeric value for a character that is a digit. For example, for the character ‘5’, the expression should evaluate to 5. Make sure you test your expression in DrJava or Eclipse. It should work for all digits ‘0’ through ‘9’.

**2C**. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Look at the complete specification of static function getNumericValue. Hey, in the first line of its specification, you see that the roman numerals are contained in type **char**! '\u216C' is 'L' –but that is different from the letter 'L', even though it looks the same. On the line above, write down what you get by casting '\u216C' to **int** and casting 'L' to **int**.

**2D**. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Which is smaller, ‘a’ or ‘A’? Write the answer to the left. Find out in two ways: (1) Use static function compareTo and (2) cast both of those characters to type **int**.

**3. Class String.** Strings are sequences of characters. Open the API specification for class String. There are several complexities in the String API:

* As discussed above for class Character, many String methods represent characters as **int**s or **byte**s instead of **char**s. Ignore those methods.
* Many methods take arguments of type CharSequence. Every String is also a CharSequence, so that you can pass Strings to these methods. For example, you can call “hello world”.contains(“hello”) even though “hello” is a String and function contains requires a CharSequence. CharSequence is an “interface”, not a class. We will learn about interfaces in about 2 weeks.
* Some useful methods have a variable number of arguments, as indicated by the “...” in the specifications. For example, you can call String.join(“/“, “apples”, “bananas”, “pears”) or String.join(“/”, “apples”, “bananas”).
* Several methods refer to “regex”es or “regular expressions”. These are patterns that match strings, such as “[a-z]\*\.txt” (which matches strings that have any number of lower case characters, followed by “.txt”). The use of regular expressions is outside the scope of CS2110. You don’t have to learn about regular expressions.

**3A.** Read the String Method Summaries. Below, make a list of the names of methods that you might find useful (for example while doing assignment A2, which deals with functions that manipulate strings).

Before you write the answers to the questions below, be sure you make sure they are correct, using either DrJava or Eclipse.

**3B.** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Suppose **int** variablex contains 7. Above, write *two* expressions that will convert x to a String. One can use simply catenation + and no calls on String functions. Another will use a String function.

**3C.** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Write a boolean expression to determine whether a given character appears at least twice in a String. Don’t use a loop. Look for functions (at least two different ones) that tell you whether a character appears in a String and that can be used to solve this problem.

**3D.** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Write a boolean expression to determine whether a given character appears exactly twice in a String. Your answer will be a modification of your answer to 3C.

**3E.** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_. Write an expression that yields String variable s but with the first letter capitalized and all other letters in lowercase. E.g. if s contains “pEnceKaIne”, the expression will evaluate to “Pencekaine”. If s contains “ComputerScience”, the expression will evaluate to “Computerscience”. Assume that s contains at least one character.