CS 122

Exam 2

Due Monday, August 1st, at 6:10pm via email for 100 points.

How To Submit

Your submission must be comprised of **two** files:

1. `multipleChoice.txt` containing your answers to all the multiple choice questions from part 1.
2. `ExamTwo.java` containing your implementation of part 2.

> ***Please email them to me before 6:10pm to both scruz3@pace.edu and src402@nyu.edu to avoid submission issues on Classes.***

I recommend that you email me what you have, regardless of whether you are finished or not, at 6:00pm to make sure that you have at least *something* submitted.

Sections

1. **Multiple Choice** *(70 pts)*
    i. **Concepts** *(50 pts)*
    ii. **Implementation** *(20 pts)*
2. **Programming** *(30 pts)*

Multiple Choice

Concepts (2.5 pts each)

1. All Java errors are exceptions.
    i. True
    ii. True, but only in situations where the IO is not the Terminal/Command Line
    iii. True, unless the user doesn't use `try - catch`
    iv. False. There are Java errors that are not considered exceptions
    v. False, unless the user creates a custom class to handle these non-exception errors

2. JavaFX considers the origin to be...
    i. 0.0, 0.0
    ii. The top left corners of the screen
    iii. A set of two non-zero positive values
    iv. Options 1, 2, and 3

v. Options 1 and 2

3. Algorithms that utilise `for` - and `while` -loops are, unlike recursive algorithms, known as...
   i. Inefficient algorithms
   ii. Incursive algorithms
   iii. Soft algorithms
   iv. Selection algorithms
   v. Iterative algorithms

4. When designing software, the programmer should...
   i. Assume that the user will always enter the correct input; all errors are the fault of the programmer
   ii. Assume that the hardware running this piece of software will never fail, so only user-caused errors should be handled
   iii. Assume that both the user and the hardware will never fail at the same time
   iv. Assume that any situation that can go wrong will, at some point, go wrong—so every error that can be handled should be handled
   v. Consider errors to be rare

5. If a JavaFX group contains a square, a circle, and an ellipse with their centres in the same location, which of the three shapes would appear as being "on top" to the user?
   i. The square
   ii. The circle
   iii. The ellipse
   iv. Neither; an exception will be raised
   v. It depends on the order in which the shapes were placed inside the group

6. Recursive algorithms have two steps—their base case, and their recursive step. The base case is...
   i. The step where the algorithm will stop calling itself and go back the call stack
   ii. The step where the algorithm will return the final answer
   iii. The step where the algorithm will call itself a final time
   iv. The step where the algorithm will throw a `RecursionException` in order to signal the end of the recursive step
   v. Options 1, 2, and 4

7. If an exception is raised at any point during the execution of a Java program, which of these is not a valid way of catching and handling the exception?
   i. By ignoring the error and hoping that it never happens
   ii. By asking the Oracle servers to scan our code to catch and handle any potential error
   iii. By catching and handling it within the method that the exception occurred
   iv. By catching and handling it within a method higher in the method hierarchy
   v. All of these are valid ways of handling errors

8. In order to translate multiple JavaFX shapes at once, the user must...
   i. Place them in the same group and then only translate the group
   ii. Translate each shape individually
   iii. Create them in their desired position from the beginning

iv. Surround them in a `move - catch` block
v. Show the scene first, and drag each object to the desired location

9. How many base cases do recursive algorithms have?
    i. They can only have one
    ii. They can have either one or two
    iii. They can, in theory, have as many as the algorithm requires as long as the recursive step leads to any of them
    iv. As a rule, they don't have any. Adding base cases is discouraged

10. The `try - catch` structure can catch and handle how many errors in one single implementation?
    i. As many as the programmer includes using the pipe (`|`) character
    ii. As many as the programmer includes as long as they are or inherit from the `IOException` class
    iii. As many as the JVM can compile in one go, which depends on how much RAM the hardware has
    iv. Only one
    v. As many as the programmer includes using the pipe (`|`) character, but it's best practice handling only one exception per `try - catch` block
11. JavaFX colours must be represented using...
    i. RGB values
    ii. Static values from the `Color` class (e.g. `Color.RED`)
    iii. The `Color` class's `color()` method
    iv. Either 1, 2, or 3
    v. 1, 2, and 3 at the same time

12. Which of the following statements is true?
    i. Any iterative method can be converted into a recursive method
    ii. Recursive methods take more time and memory to execute than iterative methods
    iii. Recursive methods are *always* simpler than non-recursive methods
    iv. There is always an `if` -statement in a recursive method to check whether a base case is reached
    v. None of the above are true

13. If a type of exception can be raised during the execution of a method, but the programmer prefers the exception to be handled further up the method hierarchy, it is absolutely necessary to include the `throws` keyword in the method's signature.
    i. True. Otherwise, the method has no way of communicating with the others in the hierarchy
    ii. False. It is only necessary with checked exceptions
    iii. False. `throws` is only a convention and is not strictly necessary
    iv. True. Both checked and unchecked exceptions must absolutely be specified by the signature in order for any method in the hierarchy to catch them
    v. False. It is only necessary with `ArithmeticException` instances

14. Which of the following are not valid ways of defining an event handler?
    i. A lambda expression
    ii. A separate file that implements the `EventHandler` interface
    iii. A private inner class that implements the `EventHandler` interface

iv. A `try - catch` block that catches event handlers instead of exceptions
v. None of the above are valid ways of defining an event handler

15. What happens if one forgets to add the base case in a recursive method?
    i. In theory, the method will not run
    ii. In theory, the method will run infinitely
    iii. In theory, the method should still return a value
    iv. In theory, the method will cause the end of the world
    v. In theory, the program will raise an `IOException`

16. Attempting to open and use a `.txt` file when an `.mp3` file was expected will lead to...
    i. A heavy fine of $1,000,100.00
    ii. An `ArithmeticException` being raised
    iii. The program crashing no matter what
    iv. An `IOException` being raised
    v. The `.mp3` file being opened as a `.txt` file with its contents rendered in binary

17. Recursive programs cannot contain `try - catch` blocks because...
    i. The method call will be too large for the JVM to keep up with it
    ii. This is untrue. There is no reason why recursive programs can't handle exceptions
    iii. It will make them unnecessarily slow
    iv. Methods that call themselves cannot contain the `throws` keyword
    v. Recursive methods can't perform actions that raise any exceptions, therefore, there is no need for them

18. What should a programmer do if their program contains a dangerous situation which is not covered by Java's pre-existing classes?
    i. Create their own exception by inheriting from the `Error` class
    ii. Create their own exception by implementing the `Throwable` interface
    iii. Create their own exception by inheriting the `Throwable` class
    iv. Throw the `Exception` class in these situations—nothing more specific should be done

19. A `TextField` fires an action event when:
    i. Clicking the `A` key
    ii. A mouse click is registered
    iii. The user types something into it
    iv. The `Enter` key is clicked
    v. The `TextField` class does not listen for events

20. The code in the `finally` block is executed...
    i. Always
    ii. If the code inside the `try` -block doesn't fail
    iii. If the code inside the `try` -block fails
    iv. If a `FinallyException` is caught and handled
    v. Never

Implementation (4 pts each)

1. Consider the following code:

```
try {
    int numerator = 10;
    int denominator = 0;
    int result = numerator / denominator;

    System.out.println(result);
} catch (Exception exception) {
    System.out.println("Exception was raised");
} catch (ArithmeticException arithmeticException) {
    System.out.println("Arithmetic exception raised");
}
```

What will be printed onto the console?

i. `0`
ii. `Exception was raised`
iii. `Arithmetic exception raised`
iv. An unhandled error message
v. Nothing

2. Consider the following code:

```java
public static void someMethod(int number) {
    if (number < 1) {
        System.out.println("STOP.");
        return;
    }

    System.out.println(number);
    someMethod( number: number / 2);
}

public static void main(String[] args) {
    someMethod( number: 10);
}
```

What will be printed onto the console when the `main()` method is called?

i. Nothing, there will be an error
ii. `STOP`
iii. `10 , 5 , 2 , 1 , STOP.`
iv. `1 , 5 , 2 , 10 , STOP.`
v. `1 , 5 , 2 , 10`

3. Consider the following JavaFX code:

```java
text.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent actionEvent) {
        System.out.println("This will not be printed.");
    }
});
```

What is the appropriate way to express this same code with a lambda expression?

i. `text.setOnAction(actionEvent -> System.out.println("This will not be printed."));`

ii. `text.setOnAction(handle(ActionEvent actionEvent) == System.out.println("This will not be printed."));`
iii. `text.setOnAction(this::processEvent);`
iv. It cannot be converted into a lambda expression

4. Consider the following code:

```java
public static int someMethod(int numerator, String denominatorString) {
    int denominator = Integer.parseInt(denominatorString);
    return numerator / denominator;
}

public static void main(String[] args) {
    try {
        int value = 0;
        String[] range = { "1", "2.", "0", "4" };
        for (String number : range) {
            value += someMethod( numerator: 12, number);
            System.out.println(value);
        }
    } catch (ArithmeticException | NumberFormatException exception) {
        System.out.println(exception);
    }
}
```

How many lines of code will be printed (not counting `Process finished with exit code 0`)?

i. One line
ii. Two lines
iii. Three lines
iv. Four lines
v. However long the error message is

5. Consider the following code:

```java
public static void someMethod() {
    try{
        throw new NullPointerException();
    } catch (NullPointerException nullPointerException) {
        someMethod();
    }
}
```

How many times will this method run before throwing a `NullPointerException` if called?

i. Once
ii. At least twice
iii. An infinite amount of times
iv. It depends on how many times your computer runs it before running out of RAM
v. The code doesn't run at all

Programming

Write a **recursive** method that will count the amount of times that a given character appears in a string. This method must have the following signature:

```
public static int countCharacters(String string, char character)
```

That is, we are returning the amount of times that `character` appears in `string`. So, for example, if we ran the following code:

```
String string = "MOS MCS 6502 1975";
System.out.println(countCharacters(string, '5'));
```

We would see the following output in our console:

```
2
```

Hints:

1. You could consider a string to simply be an array of `char` values. That is, you could apply our array head and tail strategy with a string as well (like we did with the sum of an array of numbers, for example).
2. Remember that `Strings` can be null. You should let the caller of the function know if they've passed in a null string with an appropriate exception.
3. The following `String` methods may be helpful:
   i. `charAt(int index)` : returns the character (a `char` value) at index `index` of a string.
   ii. `length()` : returns the length of a string.
   iii. `substring(int startingPoint)` : Returns a substring with all the characters of the calling string starting at index `startingPoint`. For example, if the string is equal to `"Hello"` and we print `string.substring(2)`, we will see the substring `"llo"`.

Write your method inside a class called `ExamTwo`.

Your submission must be comprised of **two** files:

1. `multipleChoice.txt` containing your answers to all the multiple choice questions from part 1.
2. `ExamTwo.java` containing your implementation of part 2.

> ***Please email them to me before 6:10pm to both scruz3@pace.edu and src402@nyu.edu to avoid submission issues on Classes.***

I recommend that you email me what you have, regardless of whether you are finished or not, at 6:00pm to make sure that you have at least *something* submitted.