Grade 12 Assignment #1 Java Programming

Math Magic

Let's practice the Java syntax you just learned.

In this project, you will become a **mathemagician** and write a small program that uses math to perform a mathematical ***magic trick***. The magic trick will involve performing arithmetic operations on any integer that you choose.

The instructions provided are general guidelines. Upon completion of the project, feel free to explore the code on your own.

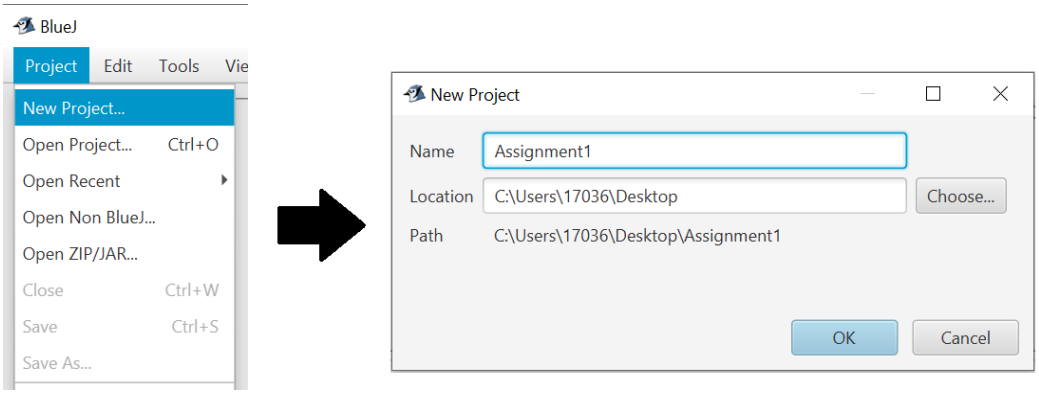
Tasks:

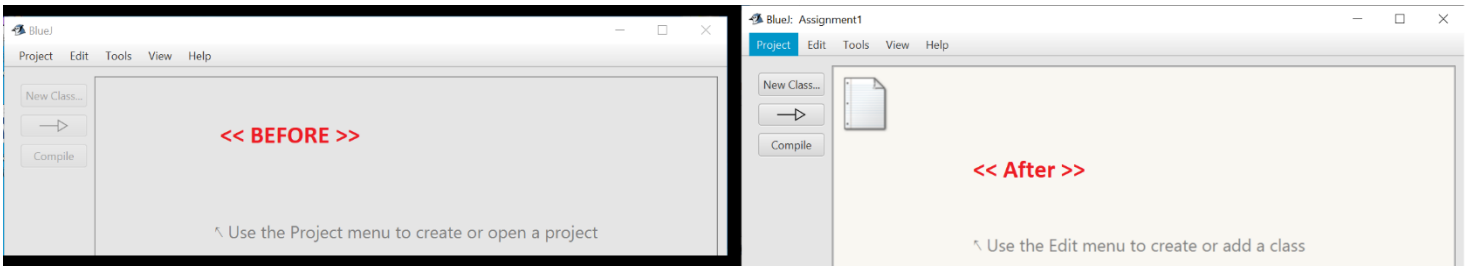
There are **20** Tasks to complete for this assignment:

← Part I: Setting Up the Project →

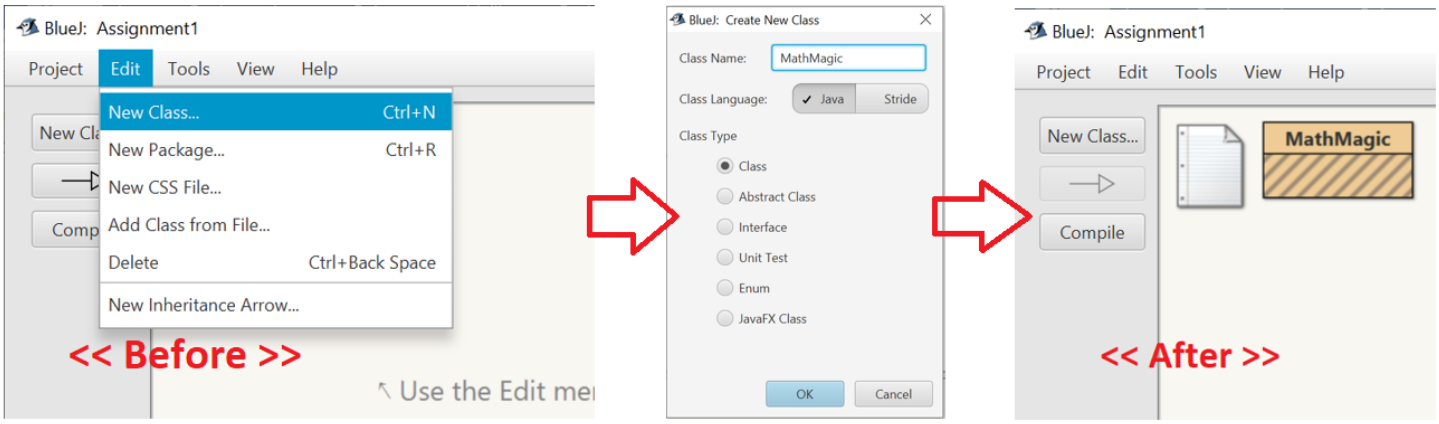
1. Open the BlueJ IDE editor and Create a "NEW Project" (Look under the Project Tab). Label the project (AKA: name the folder) "Assignment1". This folder will contain all the files necessary for the project to work/run.

   NOTE: Every Project (AKA: your Assignments) in JAVA needs to be set up in a New Folder. These systems of folders are one way that JAVA protects its programs (Scope & Encapsulation).





2. Once your New Project Folder "Assignment1" has been created, then you will need to create the file that will be used to write your program / code. Program files are called "Classes". Thus, **Create the Class** needed for the program called "Magic Math" (Look under the Edit Tab or press the Class button on the menu sidebar).



NOTE: The Class type you will be using is called "CLASS", the Filename requires a Capital letter to start and follows camel-case convention. DO NOT USE the other options from the menu: Abstract, Interface, Unit Test, Enum, JavaFx

NOTE: JAVA is an object-orientated programming language and uses the class system. Classes are used to represent objects and object names must be capitalized. Everything in JAVA must be instantiated as a CLASS, which is another way that JAVA protects its programs (Scope & Encapsulation).

3. Open the MathMagic file, look at how it is set up. There are several signatures you need to recognize.
    - **public class MathMagic { }**
        - this is the class signature in JAVA and represents the limits of the program file. There can only be 1 master class per file and it is used for designating global (not local) variables
    - **public MathMagic { }**
        - this is the constructor signature in JAVA, a constructor is used to imbue special characteristics to an object (each instance of an object created)
    - **public int sampleMethod(int y)**
        - this is a method signature in JAVA, a method is a function that can be called on an instance of an object.
        - Objects are dynamic, can create ever changing and unique code for each instance of the object (customizable)
    - **public static int sampleFunction(int y)**
        - this is a function signature in JAVA, a function is static, meaning only 1 universal outcome can be made.
    - **public static void main (String[] args)**
        - this is not in the file, but I want to show you it as you will have to create a main in the next step.
    - Note all the comments, you will need to personalize these before submitting your assignment. This will talked about below in more detail.

4. Now create a main body for the program code to reside in. The signature for a main is: public static void main(String[] args).

    NOTE: you can rewrite the sample method given to you as you probably wont use it in this assignment. I.E: change public int sampleMethod(int y) into public static void main(String[] args).

    NOTE: This is a special JAVA signature. There can only be one **main** in **ALL** the files contained in a project (folder). This MAIN represents the point of entry or starting point to launch a project and run all its files. This is similar to JavaScript where it is the index.html that represents the point of entry, the only difference is that everything in JavaScript is optional, whereas in JAVA, nothing is optional.

← Part II: Project Code →

The code you write represents the bulk of your mark. We write code to perform a task. This task is based on a magic trick that hinges on a set of equations to perform that trick. This trick is capable of changing any inputted number into 3. This project will require a familiarity of variables, object types, and arithmetic operators at the least.

5. Within the main body, create an int variable called myNumber and set it equal to any integer you like. We will refer to this integer as the **original** number from now on.

   NOTE: Main Body refers to the code with the curly braces identified by the **public static void main (String[] args)** Java Signature.

6. Next, create an int variable called stepOne and set it equal to the original number (myNumber) multiplied by itself.

7. Create an int variable called stepTwo and set it equal to the previous result (stepOne) plus the original number (myNumber).

8. Next, create an int variable called stepThree and set it equal to the previous result divided by the original number.

9. Create an int variable called stepFour and set it equal to the previous result plus 17.

10. Next, create an int variable called stepFive and set it equal to the previous result minus the original number.

11. Now create an int variable called stepSix and set it equal to the previous result divided by 6.

NOTE: Remember, the steps are important in creating the user interface. To help with the presentation thru mis-direction.

← Part III: Project Testing →

The console or Terminal Window is a coders best friend, it represents the front line and is used for debugging. The easiest way to test out a program is to log statements to the console. In Java, this is done using System.out.println(x).

12. Finally, use System.out.println(stepSix) to print out the value of the last step. What number prints to the console? This number will be printed to the console no matter what integer you choose as the original number!

NOTE: System.out.println(x) should be used regularly to check out your progress, otherwise you will have a hard time tracking down the errors / figuring out how to fix it. You should use a System.out.println(x) for every section of code written.

NOTE: Comments can be used to omit sections of code, this helps to narrow down the root of the problem

← Part IV: User Interface (Console: Terminal Window) →

The User Interface is of utmost importance when developing a program and requires a different skillset than that needed to write the actual code that makes the program work. Just remember, **a program (or answer) without context is meaningless**. Your User Interface is built with System.out.println(x) and needs the following:

13. Building an Interface

- Title (Company, Busies, Program, etc…)
- Welcome MSG (including Program Title)
- Program Introduction, talk about the goal? what to do? how to use? what to expect?
- Program Explanation (Code Transformation), explain a little about how the program works and why the answer is correct.
- Reiterate User Input (Confirm the Values Entered by the User)
- Program Output (Display what the User wants)
- Thank-You (Good-Bye, Come Back Soon) MSG

14. Visuals

- Overall Presentation? What does the interface look like?
- Formatting? Is it easy to read / follow as opposed to cluttered and unorganized?
- Whitespace? Is it cramped?
- Dialogue? Are the words well chosen, concise and meaningful?

← Part V: Formatting →

15. Code Organization (Distinct Blocks of Code)

As you begin to code you will start to see patterns emerge and will start to be able to organize your code into distinct blocks

- You should not see similar lines of code all over the place, for example all your variables should be declared in the same place.
- A distinct block of code should only contain the necessary code for a particular task to be completed.

16. Whitespace makes reading code easier

- Add 4-5 lines of whitespace between distinct blocks of code
- Add 1-2 lines of whitespace between similar code (contained within an block)

17. Comments are Required. It is important (helpful) to describe to other developers what this small Java program does. Comments should be concise (short and to the point) and used to describe what **each** part of the program does. There are 2 types:

    a. Use multi-line comments to (/* comment in the middle of */):
    b. Use a single line comment to (//then comment):

    a) Generally, multi-line comments (/* comment in the middle of */) are used above each JAVA Signature (i.e: the Class, Construct, Function, Method and Main Definitions). Blue J starts you off with a shell for these comments:

    - The Class comment (public class MathMagic { }) is special and also requires the following:
        - Title:
        - Summary:
        - Program Element List:
    - The other signature comments (like above a function or method) generally only needs a quick summary, which usually involves the parameters / return values used
        - Remember the main is special (public static void main(String[] args)) it only needs a description as it is the entry point for the program. The main has no return (void) and the parameter (String[] args) is special.

    - The End of the Program needs a multi-line comment used for any Test Code or Notes
        - NOTES: Anything deemed important that another programmer should know about, like weird bugs or irregular outcomes. Perhaps future upgrades or work in progress.
        - Test Code: The values of any input that create unexpected or weird results. Any values or things that need to be monitored

b) Use a single line comment (//then comment) for the rest:

- Place 1 comment above every Distinct Block of Code
- Every distinct block of code requires AT LEAST on comment on the side, describing the most important aspect of that block
- Some examples of common single line comments are:
    i. Identify specific values of inputs, constants, variables, or original numbers of importance.
    ii. Identify the variable, input, output or calculation section (blocks) of code
    iii. Identify the end of or last line of code in a construct (anything that ends with a curly bracket }). Write "} End of BLAH-BLAH function" after every curly brace.

← Part VI: J-Frame →

18. Now that you have an understanding of the assignment, create a J-Frame for your project. A J-Frame is one type of GUI (Graphical User Interface) used by JAVA. To help you out many J-Frame templates have been provided for you on the Shared Documents Tab of the Portal (Class Website). Look at the templates, run them and see what kind of elements you would like to use in your J-Frame.

    NOTE: Use these templates as you wish. Just copy and paste any elements you like. These templates represent a launching point for your project's J-Frame. They are used to show you the necessary signatures, constructs (element types), syntax, keywords, formatting and layouts for creating a J-Frame.

    NOTE: The user interface chosen for a program depends on its application and users. For example, programs written for an industry (like manufacturing) to be used by employees is generally written using the Console or Terminal Window. Whereas programs written for retail or end users (paying customers) is generally written using a GUI like J-Frame.

← Part VII: Project Submission →

19. Once your program is complete, make sure to test it using the BlueJ IDE (do not submit a program that does not work).
    - You will lose 30% off the top of your mark by submitting a project that does not compile & run

20. Once your program is complete, upload (drag and drop) your assignment to the portal. Look for your name under the Assignment 1 webpage.
    - There are many files in a project folder. I only need the .java files
    - Submit the MathMagic.java file