Mini Project 1 Critique

CI101 – Programming

Contents

[What grade I am claiming 2](#_Toc505369850)

[How I ensured my program worked 3](#_Toc505369851)

[Grade D and below 3](#_Toc505369852)

[Grade C 3](#_Toc505369853)

[Grade B 3](#_Toc505369854)

[Grade A 4](#_Toc505369855)

[How could the program be improved? 5](#_Toc505369856)

[What I struggled with 5](#_Toc505369857)

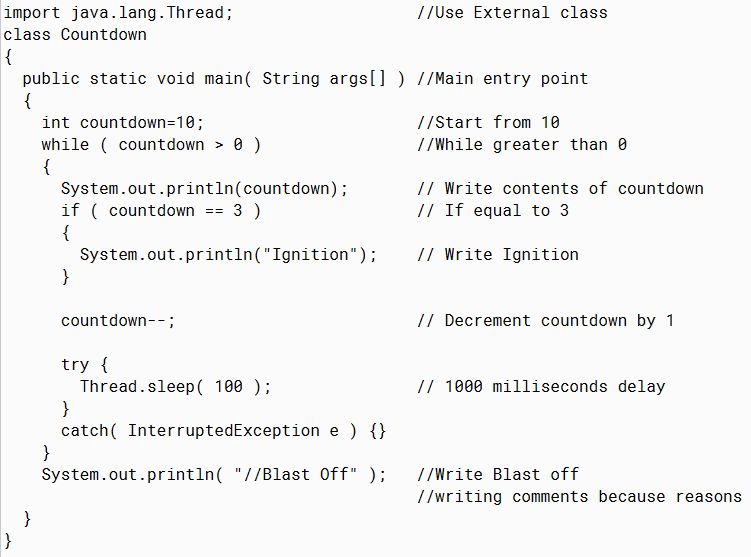
# What grade I am claiming

I feel that my submission of this mini project deserves a grade A. This is because I have fulfilled all the criteria within each grade band up to and including the A band. This can be seen in the output of my program:

Input:



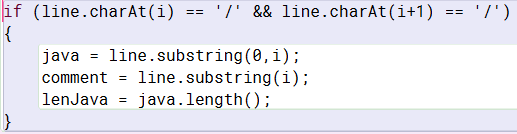
Output:

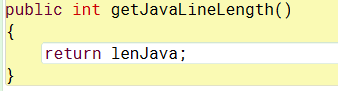


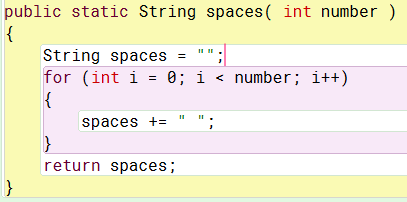
You can see that the program has indented all the comments so that all the // appear vertically underneath each other and they are all tabbed one space after the longest line. This succeeds the grades D and below. For grade C I have added lines 14 and 16 which are both empty lines of length 0 to ensure that my program could deal with that, and this fulfils grade C. I added the // in front of Blast Off on line 22 to show that my program can deal with having the // within a string, hence fulfilling the criteria for a B grade. Finally, I used .trim() function to remove all the spaces before the line, and then used for loops and if statements to add the tabbing in so that the code within {} is indented by 2 spaces more than the code outside of the braces.

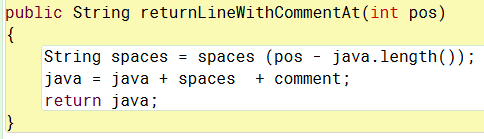
# How I ensured my program worked

## Grade D and below

I used this if statement within a for loop to split each line into the java and the comment, then used the java section with the .length() function to calculate how long each java section was.

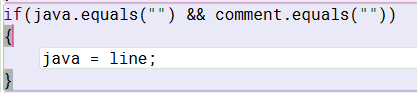
The lenJava variable was then returned in the method getJavaLineLength()

The findMaxJavaLineLength() method then used this lenJava variable from the getJavaLineLength() method to calculate the longest line length, and then set the start position of the comments to the end of the longest line +1.

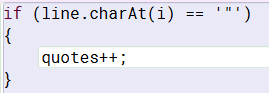
I then used this the returnLineWithCommentAt() method with the spaces() method to get the number of spaces needed to put the comment at the correct position

## Grade C

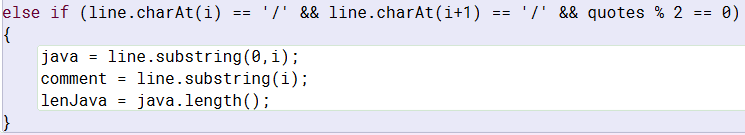
To ensure that my program could deal with lines of 0 length, I added this code which says if java is blank and the comment is blank, then set java to the value of line



## Grade B

To allow for // to occur within a string, I added a variable called quoted to track the number of quotation marks used within the program. If the number of quotation marks was odd, then the // must have been within quotation marks, and hence be within a string. If it was even, then the // would be outside of the quotation marks and not within a string.

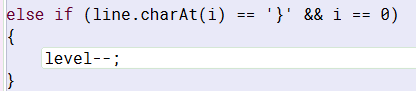
I then adjusted the if statement seen in my justification for grade D and below to account for this.



Using quotes % 2 == 0 is checking to ensure that quotes in an even number.

## Grade A

To achieve the criteria for grade A, I added a variable called level which tracked which level the code was on. I created some if statements to say that if the line started with {, then increase the level by 1. If the line started with } then decrease the level by 1. To ensure that the tabbing looked correct, I had to use a couple of if statements for the closing } and had to place them in different places.

This code checks to see if the } if the first character on the line. If it is then the level is reduced by one. This is put before the code that works out the tabbing, so this line will be indented once less than the line above

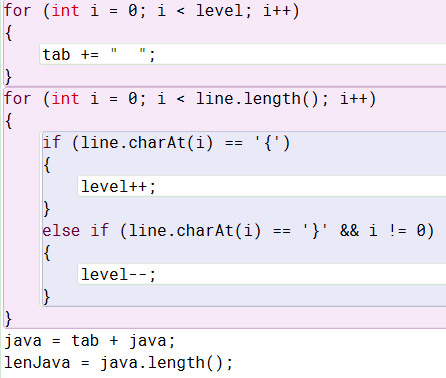
Eg

Method() Method()

{ {

Code; Instead of Code;

} }

I then wrote this code which uses a for loop to write the write amount of tabbing, adding 2 spaces for each level.

Then there is the other if statements that increase the level if there is a { on the line, and decreases the level if there is a } that isn’t at the beginning of the line. I had to do it this way to account for the {} at the end of line 20.

I then changed the java variable to add the tabbing at the beginning, and changed the lenJava variable to account for the tabbing.

# How could the program be improved?

If I was to improve this program, I would look into adding an input so that the code can already be running before you input the name of the file you want to indent. This way, I could use a while loop to keep the program running and indent multiple files at once instead of having to restart the program to indent another file.

# What I struggled with

It took me a while to work out how to get the tabbing to work. I started by trying to add the tabs to the line within the indentProgram() method within the Program.java file but I couldn’t find a way to get this to work. I then managed to get it working within the JavaLine() method within the JavaLine.java file. This is when I encountered my next issue. This was the level system.

Initially, the { character would increase the level and be tabbed more than was meant to

Eg.

Method() Method()

{ {

Code; Instead of Code;

} }

To solve this, I moved the check for the { character to after the line where the tabbing was implemented. This meant that the level variable would be increased by one after the tabbing for this line was written.

I also had to check to decrease the level before hand when the } character is present, but only if it was the first character on the line. I then had to do another check for the } character after for if it wasn’t the first character on the line. This is because the line:

catch( InterruptedException e ) {}

was printing a level down from what it should have been because the } was causing the level to decrease but the check for the { character was not running until after the tab was written.