Assignment 1

PRT 452 SOFTWARE ENGINEERING: PROCESS AND TOOLS

SHRIJANA BHANDARI

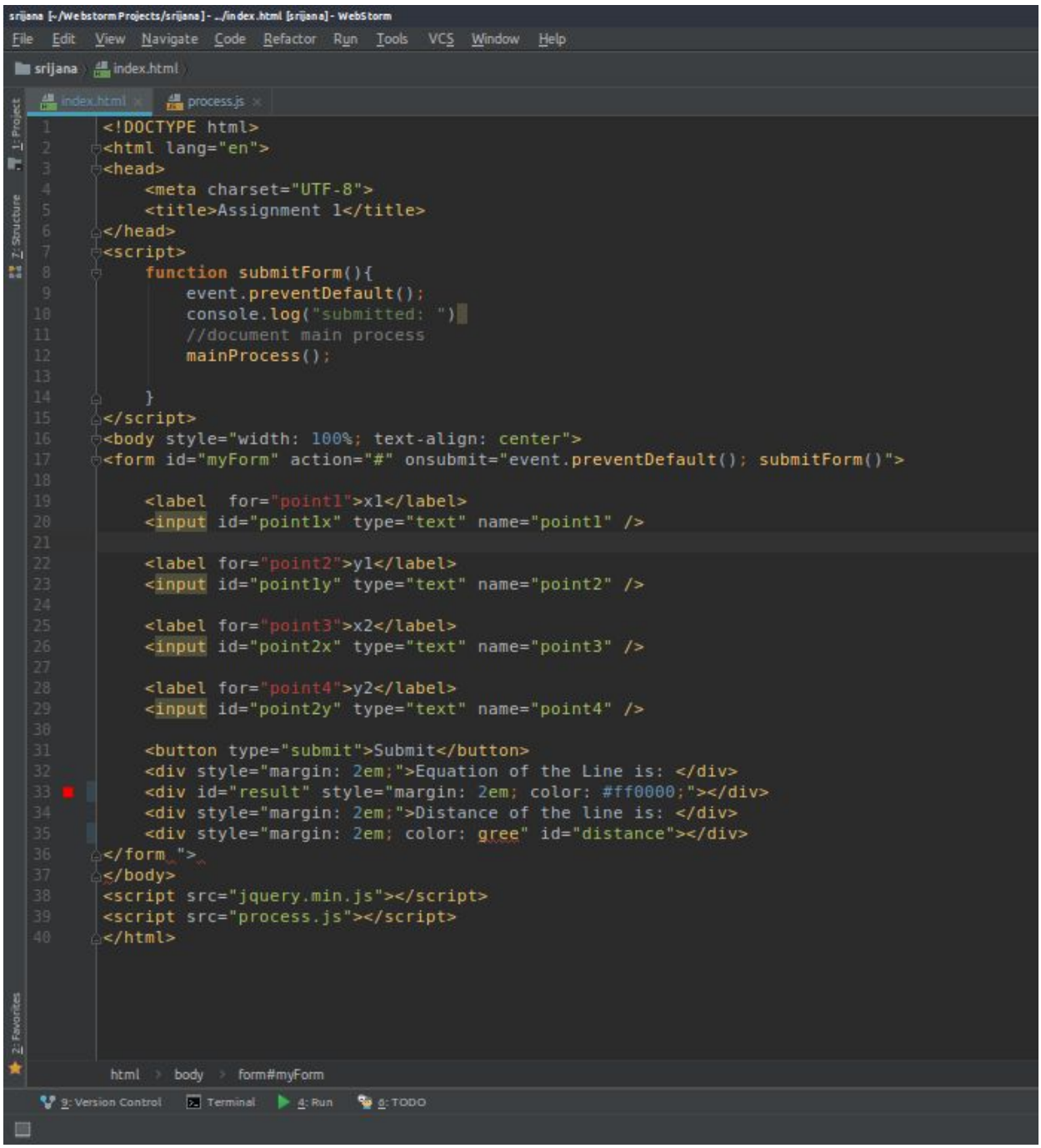
(s300775)

Question 1:

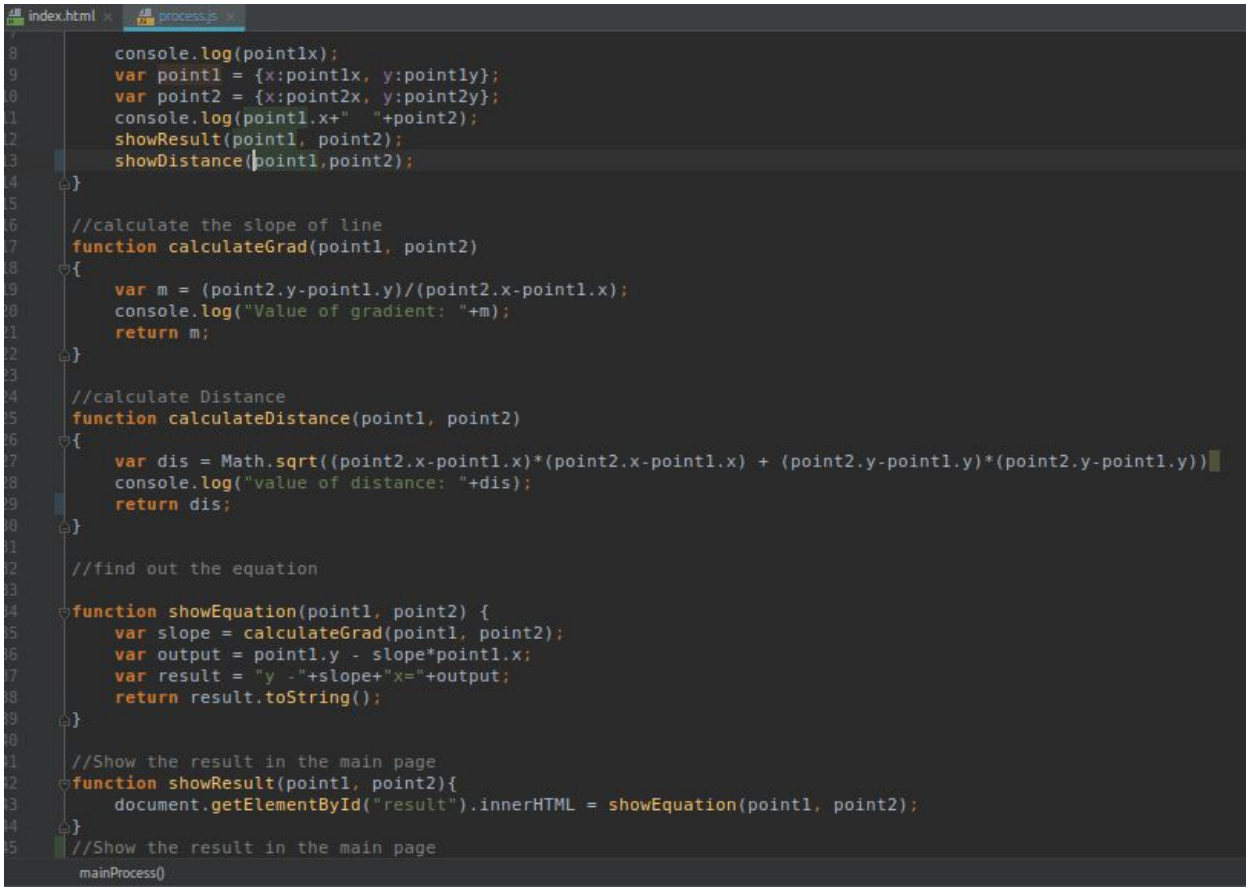
Screenshots and the steps I followed to create the program

Before refactoring.

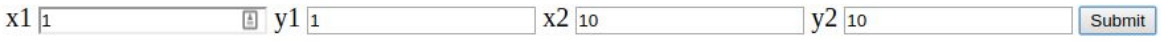
1: This is a Html file where the users can put inputs and creates a submit file.



2. In this step logic has been defined to calculate the gradient distance and equation of line.



3. Output page



Equation of the Line is:

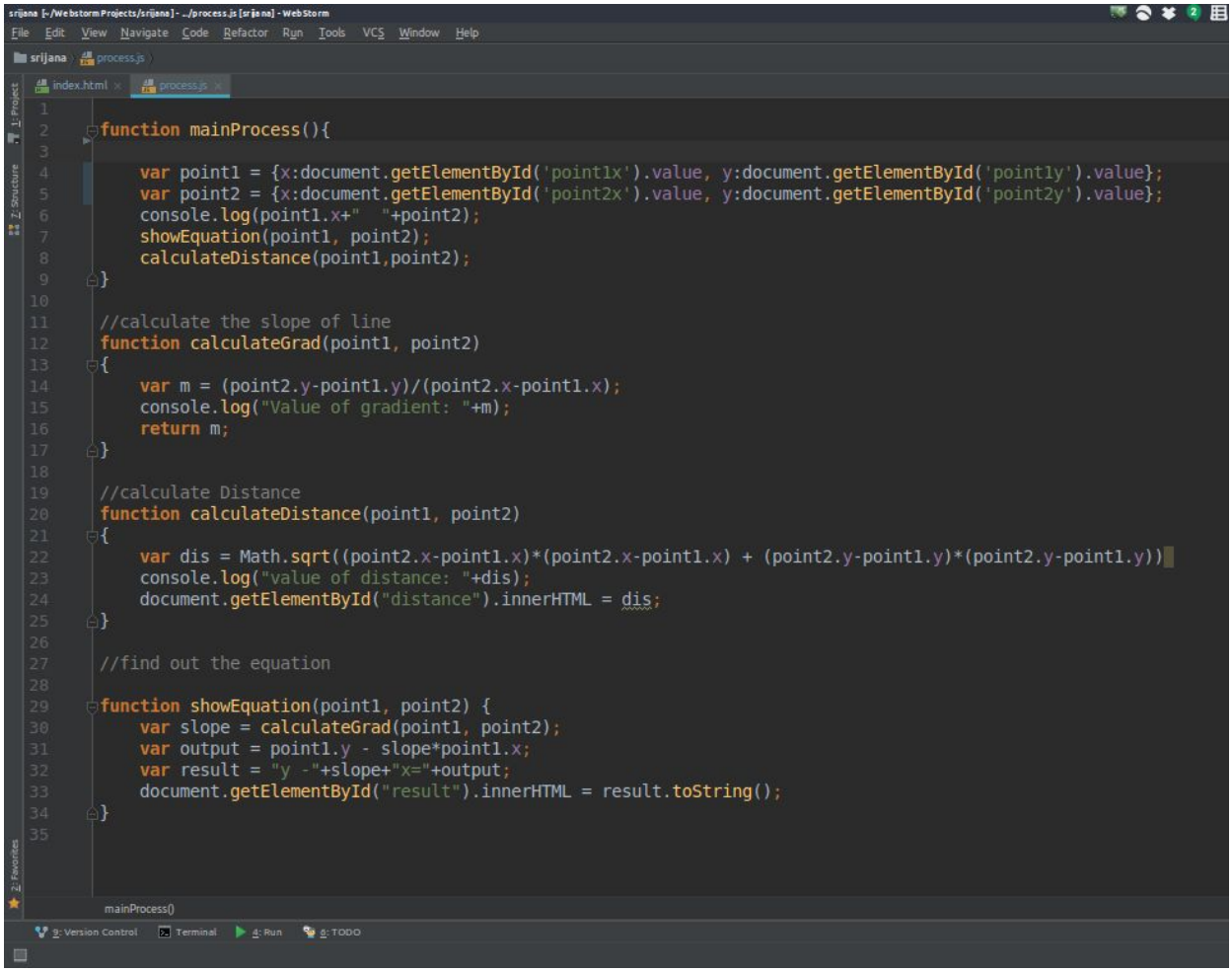
y -1x=0

Distance of the line is:

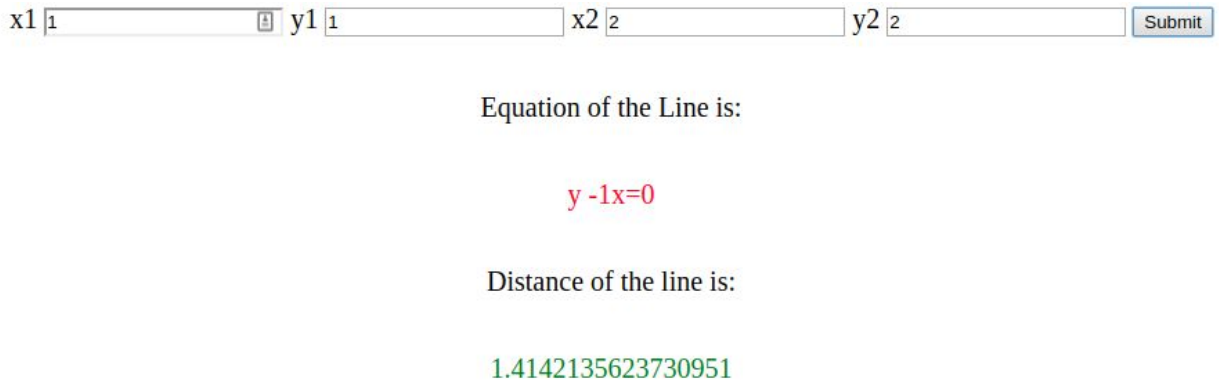
12.727922061357855

Code after refactoring:

2. Process file



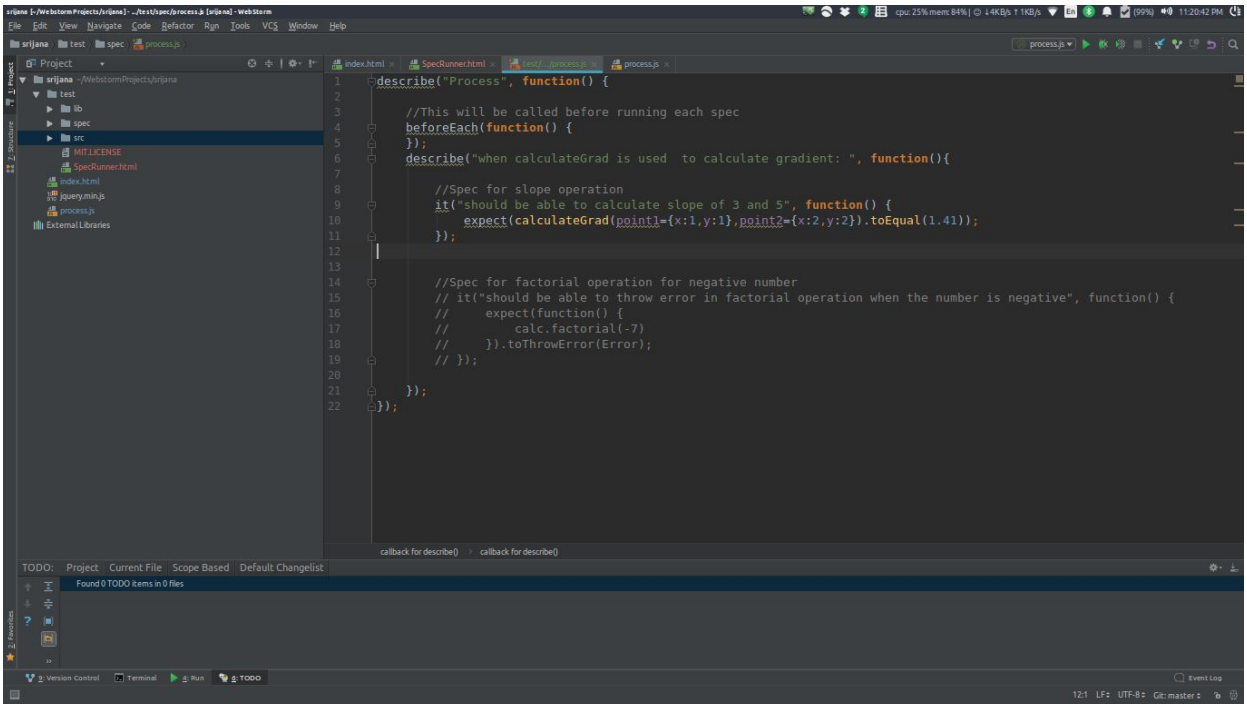
3. Output page



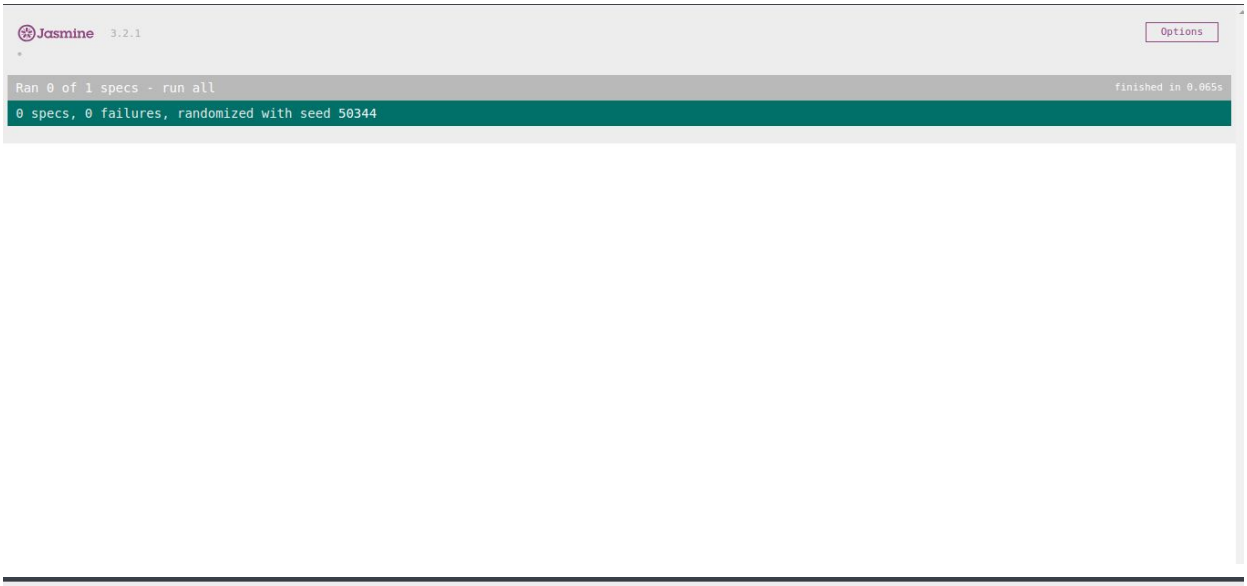
For test driven approach I have used jasmine testing library for javascript.

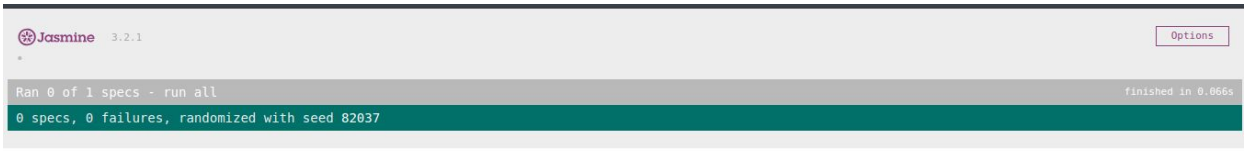
1. Failed case

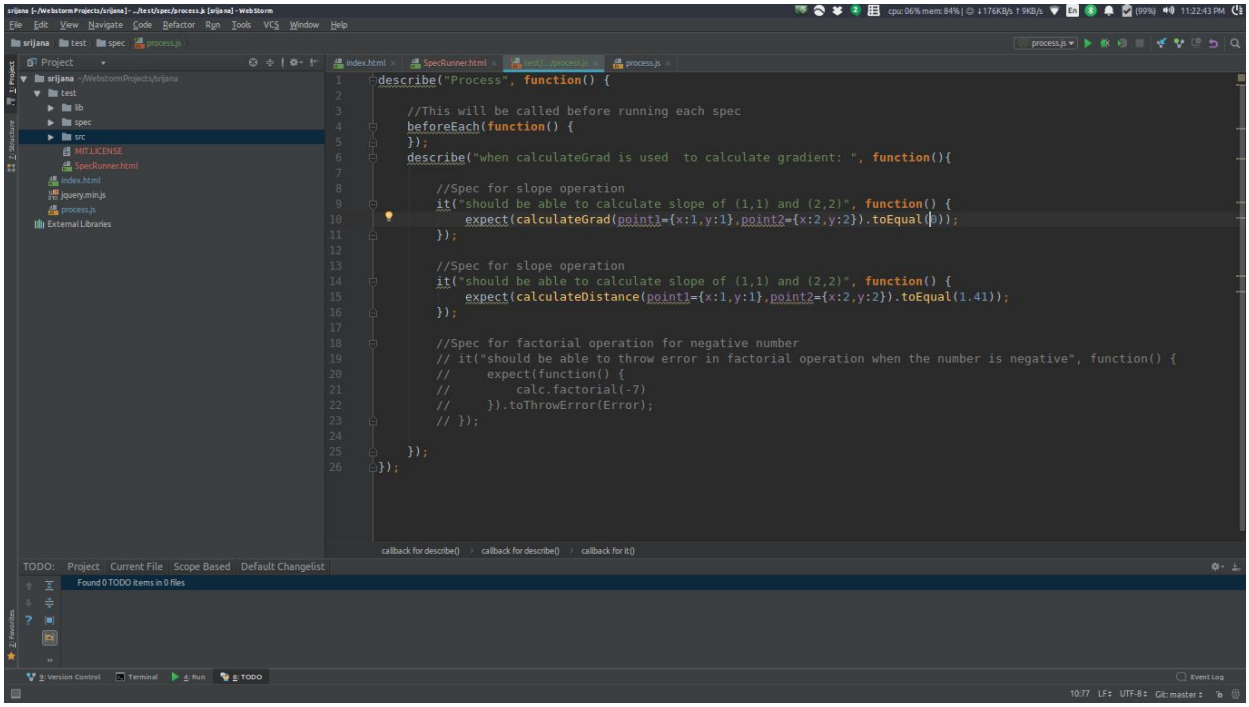
   The test was failed when wrong input point was given to the function.



After Refactoring the code, the function passed all the individual unit tests as shown in the figure below:







Question 2:

Some code smells can be listed below:

1. **Long method:** A method contains an excessive number of lines of code. For the most part, any method longer than ten lines should influence you to begin making inquiries.

   Reasons: Rationally, usually harder to make another method than to add to a current one: "Yet it's only two lines, there's no utilization in making an entire method only for that..." Which implies that a different line is included and afterward yet another, bringing forth a tangle of spaghetti code.

   Treatment: As a general guideline, in the event that you want to remark on something inside a method, you should take this code and place it in another method. Indeed, even a solitary line can and ought to be divided from into a different method, on the off chance that it requires clarifications.

2. **Data clumps:** Some of the time diverse parts of the code contain indistinguishable gatherings of variables, (for example, parameters for interfacing with a database). These clumps ought to be transformed into their own classes.

Reasons: Regularly these data groups are because of poor program structure or "copypasta programming".

Treatment: In the event that rehashing information involves the fields of a class, utilize Extract Class to move the fields to their own particular class.

3. **Large class:** A class containing numerous fields/methods/lines of code.

   Reason: Classes more often than not begin little. In any case, after some time, they get enlarged as the program develops.

   Treatment: Extract Class helps if part of the conduct of the huge class can be spun off into a different component.Extract Subclass helps if part of the conduct of the expansive class can be actualized in various ways or is utilized in uncommon cases.

4. **Duplicate code:** Two code pieces look relatively indistinguishable.

   Reasons: Duplication as a rule happens when various software engineers are taking a shot at various parts of a similar program in the meantime. Since they are taking a shot at various errands, they might be ignorant their partner has effectively composed comparable code that could be repurposed for their own needs.

   Treatments: In the event that a similar code is found in at least two techniques in a similar class: utilize Extract Method and place requires the new strategy in the two spots.

5. **Switch statements:** You have an unpredictable switch operator or arrangement of if statements.

   Reasons: Generally uncommon utilization of switch and case operators is one of the signs of object-oriented code. Regularly code for a single switch can be scattered in better places in the program.

   Treatment: To detach `switch` and place it in the correct class, you may require Extract Method and afterward Move Method.

Github Link for this project

https://github.com/shrijana17/PRT452_Assignment1

References

1. FOWLER, M., BECK, K., BRANT, J., OPDYKE, W. & ROBERTS, D. (1999) *Refactoring: Improving the Design of Existing Code*, Addison Wesley.
2. Fowler, M. (2006). *bliki: CodeSmell*. [online] martinfowler.com. Available at: https://martinfowler.com/bliki/CodeSmell.html.
3. Refactoring.guru. (2014). *Data Clumps*. [online] Available at: https://refactoring.guru/smells/data-clumps.
4. Refactoring.guru. (2014). *Message Chains*. [online] Available at: https://refactoring.guru/smells/message-chains.
5. Refactoring.guru. (2014). *Remove Middle Man*. [online] Available at: https://refactoring.guru/remove-middle-man.
6. Different kinds of code smells and how to refactor them,
7. http://www.testingeducation.org/pt/Refactoring-smells.pdf
8. How to approach code smells, journal of software: evolution and process published by john wiley & sons Ltd
9. https://onlinelibrary.wiley.com/doi/full/10.1002/smr.1886
10. What are the code smells and different what are the different approaches to clean that code smell
11. https://github.com/lee-dohm/code-smells
12. Refactoring to patterns by Joshua kerievsky, Chapter 4 code smells

13. https://www.safaribooksonline.com/library/view/refactoring-to-patterns/0321213351/ch04.html
14. Gradient, distance formulas for idea. https://www.mathsaccelerator.com/algebra/gradient