Hyperiondev

TASK

Capstone Project II — Refactoring

Visit our website

Introduction

WELCOME TO THE REFACTORING CAPSTONE PROJECT!

You've made it to the second Capstone Project for this level! Well done! Refactoring is vital to ensure quality code. In this Capstone Project, you will improve the quality of code written in your first Capstone project. You will do so by adding exception handling and refactoring your code.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

DEVELOPER PORTFOLIO

Your developer portfolio should flaunt your skills as a Developer. Prospective employers will want to see not only that you can write code to solve problems, but also that your code is of good quality. Code that is designed to deal with exceptions elegantly and that is well refactored to adhere to best-practice guidelines provided in style guides is of much better quality than code that is not written to meet these criteria! In this project, you will improve the quality of the code that you wrote in the previous Capstone Project.

REFACTORING REMINDERS

Refactoring includes doing some of the following:

- **Use meaningful names**. Throughout your code, make sure that all variables, functions, modules, classes, etc. are given meaningful, descriptive names.

- Make sure **that your code is broken down into modules** (functions, classes, methods, etc.).

- **Be as brief as possible**. If you can do something with fewer lines of code without detracting too much from the readability of your code, do so.

- **Make sure inputs to functions and output from functions are clear**. Make sure that your functions return real, meaningful output. Make input into your functions as clear as possible.

- **Remove unnecessary code**. As we code, we often create much code that we can, and should, remove from the final version of our app. Delete:
    - Variables that are declared but never used.
    - Functions that are never called.
    - Code that you commented out because you thought you might need but you don't.
    - Statements used for debugging. Your code is likely to include things like `System.Out` statements that you used to debug your system. For the final version of your app, get rid of these types of statements.

- **Ensure classes with high levels of cohesion**. Cohesion means that all the methods and properties in a class are closely related to each other and only perform one function. If a class takes ownership of an assortment of unrelated responsibilities, it should be broken up into multiple classes. Each class should only have one responsibility. All methods and data in that class should be strongly related to that responsibility.

- **Ensure programs with weak coupling**. Coupling describes how dependent classes are on each other. Weak coupling means that objects are not dependent on each other so that you can change one class without having to make changes to different classes. To enable weak coupling, we make sure that any variables/properties that should belong to an object are declared locally in that class. These properties can also be made private. Then other classes can't get to that data directly. If another class wants to change the properties of a class, they have to use the class's accessor or mutator methods to do this.

- **Remove `Set()` methods for fields that cannot be changed.** If a field is supposed to be set at object creation time and not changed afterwards, initialise that field in the object's constructor rather than providing a misleading `Set()` method.

- **Hide methods that are not intended to be used outside the class.** Do this by making these helper methods private.

- **Hide data members.** Public data members are always a bad idea as they blur the line between interface and implementation. Additionally, they inherently violate encapsulation and limit future flexibility. Consider hiding public data members behind access methods.

- **Make sure that a loop is too long or too deeply nested.** The code inside a loop can be converted into methods. This helps to refactor the code better and reduce the loop's complexity.

- **Use `break` or `return` instead of a loop control variable**. If you have a variable within a loop (like a boolean variable called 'done') that's used to control the loop exit, use `break` or `return` to exit the loop instead.

- **Return as soon as you know the answer instead of assigning a return value** within nested if-then-else statements. Code is often easiest to read and least error-prone if you exit a method as soon as you know the return

value. The alternative of setting a return value and then unwinding your way through much logic can be harder to follow.

- **Make sure that a method's parameter list doesn't have too many parameters.** Well-factored programs have many small, well-defined methods that don't need large parameter lists. A long parameter list is a warning that the abstraction of the method interface has not been well thought out.

- Comments are important but **do not use comments as a crutch to explain bad code**.

Compulsory Task

Follow these steps:

- Open the first Capstone Project that you created in this level (the object-oriented program for a structural engineering company called "Poised".)
- Enhance your program by doing the following:
    - Incorporate exception handling to your code. Your code should include at least two try-catch blocks *(Go back to the Defensive Programming task if you're not sure where to start).*
    - Make sure all your code is properly debugged. Make sure that the runtime and logical errors are also identified and corrected.
    - Fix the indentation and formatting of the code so that it adheres to the guidelines provided **here**.
    - Make sure that all the names of variables, classes, methods, etc. adhere to the guidelines provided **here**.
    - Refactor the code to improve the quality and readability of the code in other ways highlighted in this project.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

Click here to share your thoughts anonymously.

REFERENCES

McConnell, S. C. (2004). *Code Complete*.(2nd ed.). Redmond, Washington: Microsoft Press.