Source Code Refactoring Report
for
Chemistry Calculator

Course Title: Software Maintenance Lab
Course Code: SE4204

Team Members

Abdullah An-Noor (ASH1825001M)
Fazle Rabbi (ASH1825004M)
Rahat Uddin Azad (ASH1825022M)
Saifur Rahman (ASH1825031M)
Anwar Kabir Sajib (ASH1825038M)

Submitted To

Dipok Chandra Das
Assistant Professor
IIT, NSTU

Date of Submission
02-March-2023

Table of Contents

List of Tables

1. Importance of Code Refactor:

Code refactoring is the process of improving the structure, readability, and maintainability of existing code without changing its external behavior. It is an essential practice in software development projects for several reasons:

Improve Code Quality: Refactoring helps to improve the quality of code by removing duplication, reducing complexity, and improving readability. This makes it easier to understand and modify the code, which ultimately leads to fewer bugs and better performance.

Enhance Maintainability: Refactoring makes the code easier to maintain by breaking it into smaller, more manageable pieces. This allows developers to make changes more easily without introducing new bugs or breaking existing functionality.

Increase Developer Efficiency: Clean and well-structured code is easier to work with and understand, which leads to faster development times and fewer errors.

Reduce Technical Debt: Technical debt is the accumulation of shortcuts taken during the development process, which can make it difficult to maintain the codebase over time. Refactoring helps to reduce technical debt by addressing these shortcuts and improving the overall quality of the code.

Enable Code Reuse: Refactoring can make code more modular and reusable, which can save time and effort in future development projects

2. Fundamental steps for refactoring:

There are some Fundamental steps which we follow in refactoring Process of **"Chemistry calculator"** project.

1. Identify what to refactor.
2. Determine which refactoring's to apply.
3. Ensure that refactoring preserves the software's behavior.
4. Apply the refactoring's to the chosen entities.
5. Evaluate the impacts of the refactoring's.

2.1. Identify what to refactor

For the Chemistry Calculator project, the following code smells and areas for improvement have been identified:

Large and complex methods: There are several methods in the project that are too long and complex, making them difficult to understand and maintain. These methods need to be broken down into smaller and more manageable pieces.

Duplicated code: There are instances of duplicated code throughout the project, which increases the risk of bugs and makes it harder to maintain the code. These instances of duplicated code should be identified and refactored into reusable functions.

Long parameter lists: Some methods in the project have long parameter lists, which can make them difficult to use and understand. These methods should be refactored to use objects instead of long parameter lists.

Exception handling: The project has inconsistent and unclear exception handling, which can lead to unexpected behavior and errors. The exception handling code should be refactored to make it more consistent and clearer.

Primitive obsession: There are several places in the project where primitive types are used instead of objects. This can make the code harder to understand and maintain, and can lead to errors. The use of primitive types should be refactored to use objects where appropriate.

Complex conditionals: Conditionals that are too complex or have too many nested levels can be hard to understand and can lead to bugs.

Unused code: There is some unused code in the project that should be removed to simplify the code and reduce the risk of bugs.

Dispensable: Overuse of comments can indicate that the code is hard to understand and needs refactoring to make it more readable and maintainable.

Class	**Code Smells**
Home.java	Large method, Long parameter list
Sidebar	Large method, Long parameter list
TitrationPanel.java	Exception handling, Large method, Duplicated code, Complex conditionals
MolarMassPanel.java	Large method, Duplicated code, Unused code
HistoryFrame.java	Large method
Compound.java	Dispensable
CompoundManager.java	Dispensable
FxpieChart	Dispensable
Fraction.java	Complex conditionals, Large method
EquationBalance.java	Complex conditionals
DatabaseSerializer.java	Dispensable
PercentOfCompletionPanel	Large method, Duplicated code
EquationBalancePanel	Large method, Duplicated code

Table 1: Identified code smells

2.2. Determine which refactoring's to apply

There we mention which specific refactoring's we have chosen to apply identified code smells.

Refactoring ID	Refactoring name
R1	Extract Method
R2	Introduce Parameter Object
R3	Decompose Conditional
R4	Consolidate Duplicate Conditional Fragments
R5	Replace Temp with query
R6	Move Field
R7	Delete Comments
R8	Consolidate Conditional Expression
R9	Replace Nested Conditional with Guard Clauses
R10	Replace data type with object

Table 02: Refactoring Sets

Code Smells	Remedies
Large method	R1, R2
Long parameter list	R2
Exception handling	R3
Complex conditionals	R3, R4, R9
Duplicated code	R1
Unused code	R7
Primitive obsession	R10
Dispensable	R7

Table 03: Refactoring Technique mapping

2.3. Ensure that refactoring preserves the software's behavior

Two pragmatic ways of showing that refactoring preserves the software's behavior:

1. Testing: Exhaustively test the software before and after applying refactoring's, and compare the observed behavior on a test-by-test basis.
2. Verification of preservation of call sequence: Ensure that the sequence(s) of method calls are preserved in the refactored program

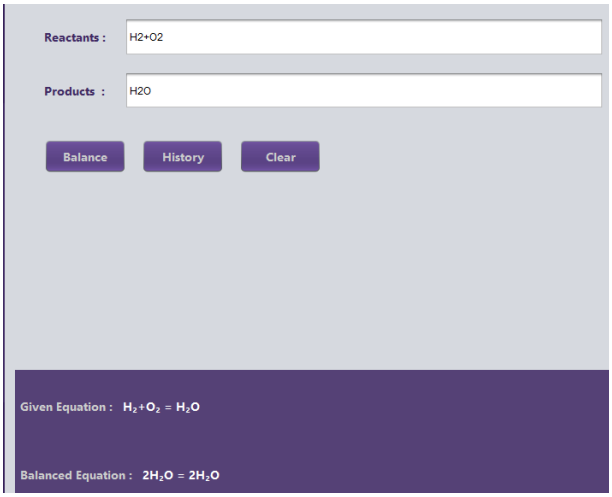
Testing:

Test No: 01

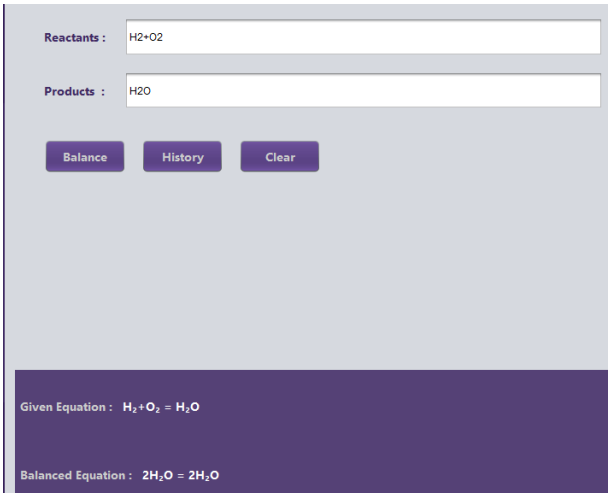
Class Name: EquationBalance.java

Test Case: System provides the balanced equation properly

Refactoring Type: Decompose Conditional



Before Refactoring



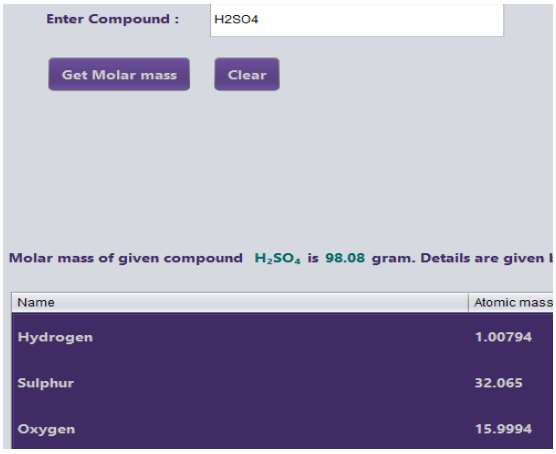
After Refactoring

Test No: 02

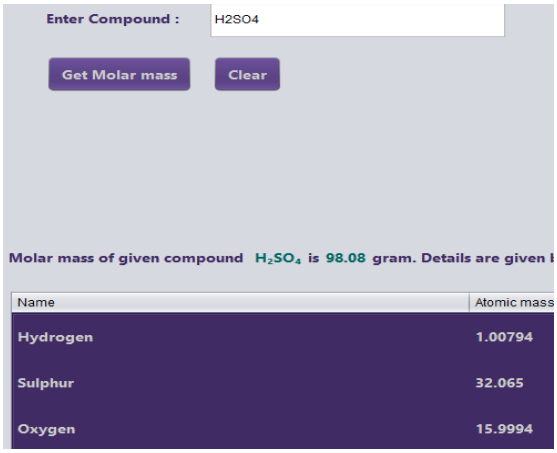
Class Name: Fraction.java

Test Case: System provides the atom molar mass fraction value

Refactoring Type: Decompose Conditional

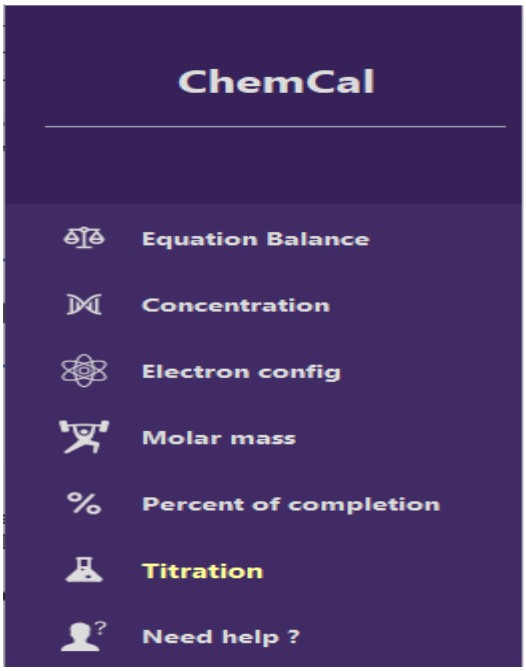


Before Refactoring

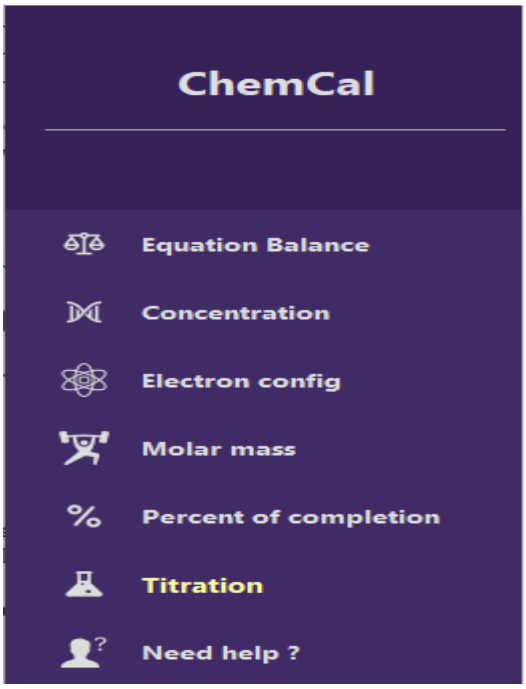


After Refactoring

Test No: 03
Class Name: Sidebar.java
Test Case: System provides a proper sidebar
Refactoring Type: Extract Method



Before Refactoring



After Refactoring

2.4. Apply the refactoring's to chosen entities

Refactoring ID	Refactoring name
R1	Extract Method
R2	Introduce Parameter Object
R3	Decompose Conditional
R4	Consolidate Duplicate Conditional Fragments
R5	Replace Temp with query
R6	Move Field
R7	Delete Comments
R8	Consolidate Conditional Expression
R9	Replace Nested Conditional with Guard Clauses
R10	Replace data type with object

Table 04: Refactoring sets

No.	Class Name	Refactoring ID
01	Compund.java	R1, R2
02	EquationBalance.java	R7, R3
03	DatabaseSerializer.java	R7
04	Fraction.java	R2
05	Sidebar.java	R1, R3, R4, R5
06	CompoundManager.java	R1
07	TitrationPanel.java	R2, R3, R4
08	Titration.java	R3, R9
09	Matrix.java	R7, R1, R3
10	MolarMassPanel.java	R1
11	PercentOfCompletionPanel	R1, R2
12	EquationBalancePanel	R1, R2
13	HistoryFrame.java	R1

Table 05: Refactoring Technique class wise mapping

2.5. Evaluate the impacts of the Refactoring's on Quality:

Class name	**Internal Qualities of the software**				
	Size	Complexity	Coupling	Cohesion	Testability
Compund.java	Decrease	Low	Loose	Increased	Yes
EquationBalance.java	Decrease	Constant	Constant	Increased	Yes
DatabaseSerializer.java	Decrease	Low	Loose	Constant	Yes
Fraction.java	Decrease	Low	Loose	Increased	Yes
Sidebar.java	Constant	Constant	Constant	Constant	Yes
CompoundManager.java	Decrease	Low	Loose	Increased	Yes
TitrationPanel.java	Constant	Constant	Constant	Constant	Yes
Titration.java	Decrease	Medium	Loose	Increased	Yes

Table 06: impacts of the Refactoring's in Internal Qualities

Class name	**External Qualities of the software**				
	Performance	Reusability	Maintainability	Scalability	Extensibility
Compund.java	Constant	Reusable	Maintainable	Constant	Extensible
EquationBalance.java	Increased	Reusable	Maintainable	Scalable	Extensible
DatabaseSerializer.java	Constant	Constant	Constant	Constant	Constant
Fraction.java	Increased	Reusable	Maintainable	Constant	Extensible
Sidebar.java	Constant	Constant	Maintainable	Scalable	Extensible
CompoundManager.java	Constant	Reusable	Maintainable	Scalable	Extensible
TitrationPanel.java	Constant	Constant	Constant	Constant	Constant
Titration.java	Increased	Reusable	Maintainable	Scalable	Extensible

Table 07: impacts of the Refactoring's in External Qualities

Github Link: https://github.com/minionsrahat/SoftwareMaintenanceLab_Refactoring