A Comprehensive Review of Refactoring Techniques

[1]Sukhdeep Kaur , [2] Dr. Raman Maini

Department of Computer Engineering,
Punjabi University, Patiala, India

Abstract: **Refactoring is a crucial process to improve the quality of software. Refactoring is part of software engineering that improves more readability of program and maintainability of the software. Refactoring is a most widely used technique that gives the code simpler, cleaner, reusable, extendable, maintainable or other characteristics by transforming a program. In programming languages, bad smell or code smell is a code or design problem that makes the software hard to understand and maintain the code. Basically, bad smells are structured characteristics of software that indicate a problem, may be need it refactoring of code. In this paper some refactoring techniques discussed that are used to remove code smells from code or program. Simulation has been done for some refactoring techniques like Rename method, Extract method, Move method**, **P**ull **up m**ethod **using Eclipse tool. Eclipse refactoring tool supports the java development language. It has been observed that pull up method is better as compared to other refactoring methods because pull up method dispose the complexity and duplicate code from program.**

Keywords: Software Refactoring, Bad Smell, Refactoring Tool, Refactoring technique, Result using Eclipse.

I. INTRODUCTION

Refactoring is a disciplined technique used for restructuring an existing code of program and altering its internal structure without changing its external behavior, in case of complexity in the program. According to Martin Fowler "Using refactoring, observable behavior is not changed if it alter the internal structure of software for easier to understand and cheaper to modify the code" [1]. When refactoring is used to modify the software, it improves readability, maintainability, and extensibility of the code. Refactoring is an important part of software development cycle and refactoring tools are also critical to make refactoring fast and behavior preserving. Basically, refactoring is a process to change a software system in such a way that remains the same result, only improves the internal code. For various software applications, refactoring techniques are applied to artifacts like Models, Unified Modeling Language (UML), Databases, HTML (source code) documents. In programming language, it provided advantages like easy to understand, reduce duplicate code and inconsistent code. The goal of refactoring is to make code easier to maintain and reusable in the future [2]. Refactoring techniques are required for adding new features, fix a bug/bad smells and for greater understanding.

There are various steps used to refactor the code [3].

1. Apply unit test to program code.
2. Find some code that have "smell" in program.
3. Determine how to simplify these code smells.
4. Select and apply refactoring technique for removing the code smell.
5. Repeat the simplify/test cycle until the smell is gone from code.

II. BAD SMELL

If it stinks, change it. Pieces of code that is wrong in some sense and ugly to see [4]. Bad smells in other words, it is a code or design problem that occurs when structural characteristics of software are defined but it does not produces any error at the execution time. This problem makes the software hard to maintain the code and may be it require active the refactoring of code to remove the bad smells. [5].

2.1. Bad Smells in Code: The bad smells in the code are defined as follows [6]-[7].

2.1.1. Duplicate Code
When same code elements exist in multiple places rather than one place, it is a duplicate code.

Duplicate code become badly because if you modify one instance of duplicated code but not the others, you may have introduced a bug. E.g. Having the same expression in two sibling subclasses.

2.1.2. Long Method
Many statements, loops, variables in a one method is a long method. Long Method is too long, so it provides difficulty to understand the code. E.g. Over 20 lines of method are usually a bad sign. Fewer than 10 lines of method are typically good.

2.1.3. Large Class
A class that is trying to do too much function is a large class. Large class reduce the cohesion from code because it has many instance variables or methods. E.g. More than seven or eight variables indicate a bad sign.

2.1.4. Feature Envy
A method that is define in one class but it is more interested in the attribute of other class where it is presently located and focus of the envy in feature envy is the data. Method in one class seems happier in other class. E.g. Finding a (part of a) method that makes heavy use of data and methods from another class.

2.1.5. Long Parameter List

Many parameters passed into one method. It is hard to understand and become with inconsistent code. E.g. public void marry (String name1, Int age1, Boolean gender1, String name2, Int age2, Boolean gender2){ ……. }

2.1.7. Data Class

If getter/setter methods, variables and properties for fields in only one class then it is a data class.

III. TOOLS

Refactoring support various tools like

3.1. Smalltalk Refactoring Browser

The first widely used refactoring tool was Smalltalk refactoring browser that used for implementing most of the standard classes, methods and fields refactoring for the Smalltalk language [8]. Smalltalk is a very neat and clean language which has more automatic processing than other languages such as C++.

3.2. Eclipse

Eclipse tool provided by IBM that is an open source development environment and it creates a 'universal tool platform'. At the core, it is a language-independent development area which offers some refactoring features like Rename variable, parameter, Move method, Extract method, Pull up Method, Extract class etc. Eclipse tool is interesting because it aims to be a general development environment, rather than focusing on Java [8].

3.3. Idea

Idea is a commercial development environment created by IntelliJ. JetBrains creates the IntelliJ, Idea is a integrated development environment in java. Idea has the refactoring features like Rename Method for package, class, field, method parameter and local variable, Extract Method, Introduce Variable. Idea does not implement many refactoring techniques, for this reason it could not be described as a refactoring browser.

IV. SIMULATION

Using eclipse tool provide simulation for some refactoring techniques. Eclipse tool has most powerful group of refactoring techniques to enhance the productivity of software [8]. Eclipse tool defined with some source code examples in java language for what is the purpose and use of each refactoring technique.

In eclipse tool, refactoring can be grouped into three parts [9]:

a) Change the name and internal structure of code that include rename elements, classes, variables, interfaces and move methods and classes e.g. rename method.

b) Change the internal code within a class code that include fragment of a code into a new separate method from selected code in a method and getter/setter methods are generated for fields e.g. extract class, extract method.

c) Change the internal code at the class level that include push down the methods or fields from a super class to a subclass, pull up the methods or fields from subclass to a super class and move method from one class to another class e.g. pull up method, push down method, move method.

Simulations for some refactoring methods are defined as follows:

1. Rename Method: This is simple refactoring used to rename a methods, method parameters, fields, local variables, compilation units, packages, source folders, projects, etc.

If the name of any elements in a code like variable name does not reveal out its purpose why to use then change the name of its variable.

In the figure1 (a) the name of class is pp before refactoring. Class pp name have does not sense for understanding what work is done by that class.

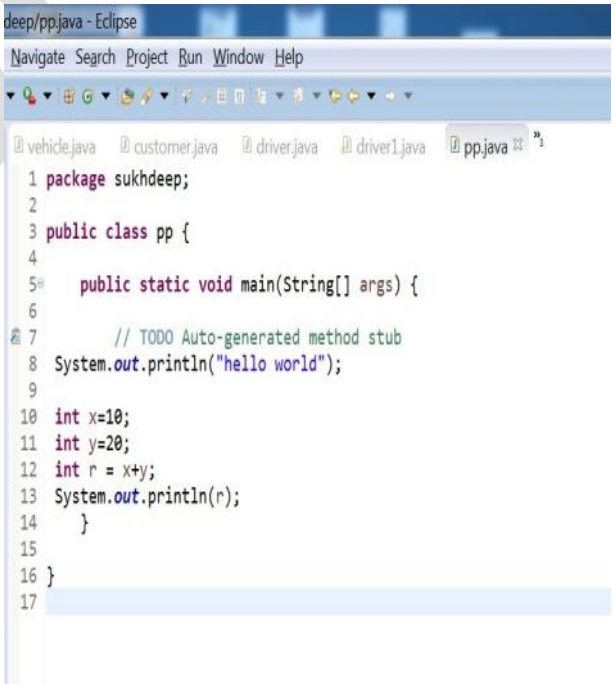

Figure 1(a) code before rename refactoring

So change the name of class pp using rename refactoring method shown in figure1 (b). For this select Refactor>Rename option and type the name of class simpletest and click on Finish.

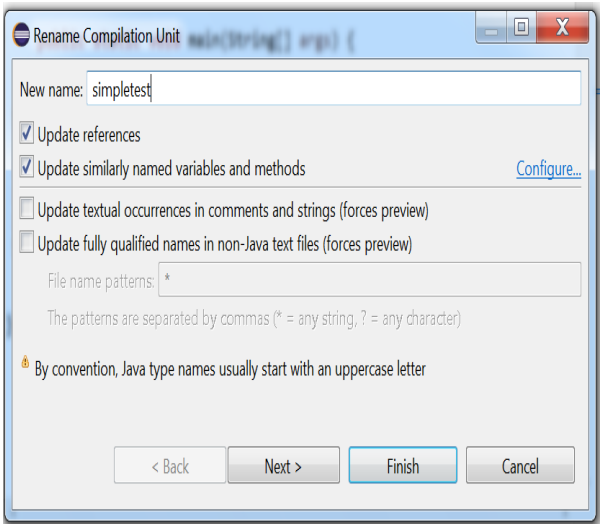

Figure 1(b) select option and change the name of class

After select the refactor option, type the name of class is simpltest and result show in figure1 (c).

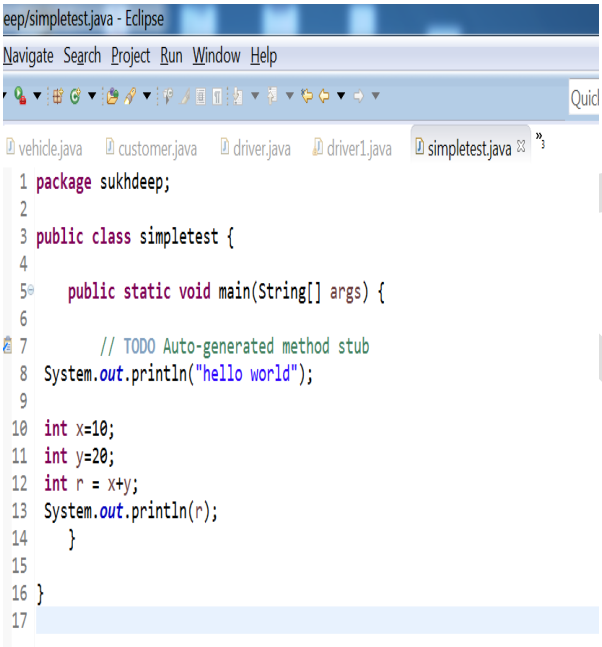

Figure1(c) code after rename refactoring

2. Extract Method: Extract Method is typically used when a method is too long. It contains fragment of long method into a new separate method that defined the purpose of the method what for using. Extract method create a new method from currently selected statements or expression in code and replaces this selection with a reference to the new method. Extracting the relevant code out into its own method allows it to be called elsewhere, and makes the original method easier to read and understand. In Figure2 (a) print the marks of class student. For effective way to understand the code, apply extract method.

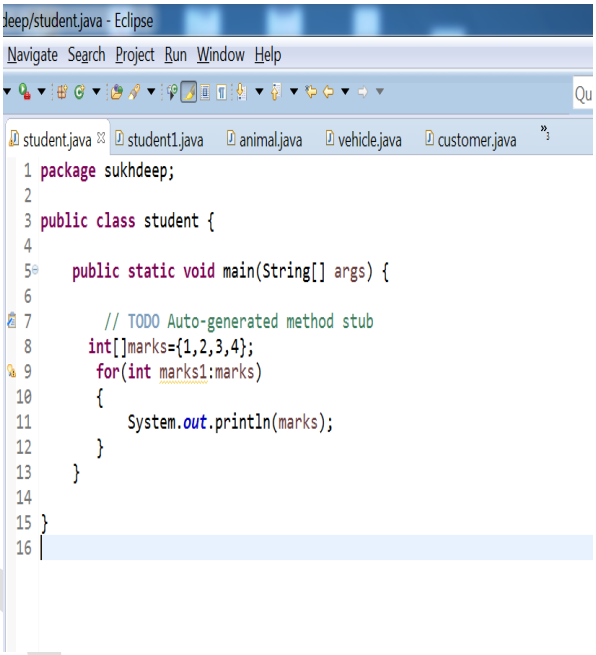

Figure 2(a) code before extract refactoring

Select Refactor>Extract Method option and type the name of method printmarks and select access modifier. In figure 2(b) public access modifier is selected and click on Ok.

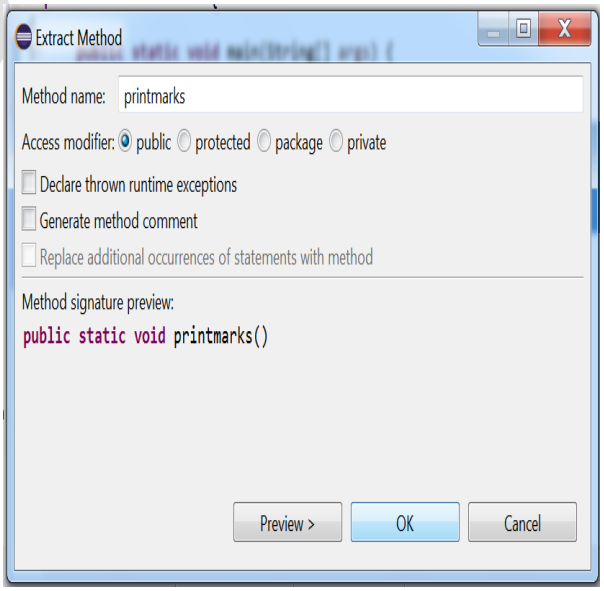

Figure 2(b) select option and type name of method

After refactoring result show in figure2 (c) . It creates new method printmarks that defined the better understanding to read the code as compare to before refactoring the code.

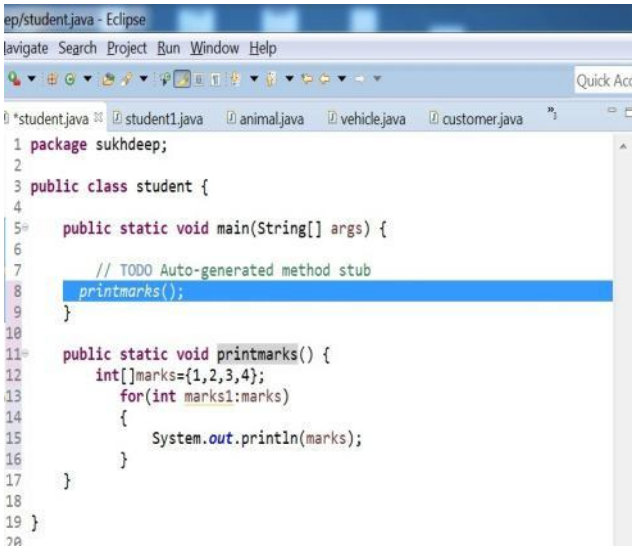

Figure 2(c) code after extract refactoring

3. Move Method: If a method define in one class but it is more used by another class where it is currently located, then move that method to the another class [10]. In the figure3 (a) driver class have method warehouse that method is mostly used by driver1 class. So move this method to driver1 class.

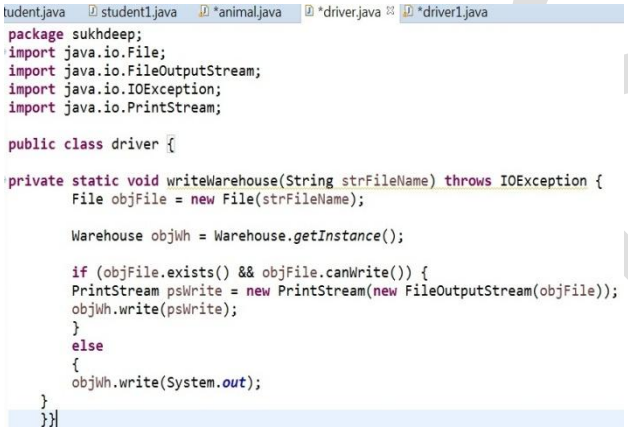

Figure 3(a) code before move refactoring

For moving select Refactor>Move option and type the name of class destination where to move the method, show in the figure3 (b)

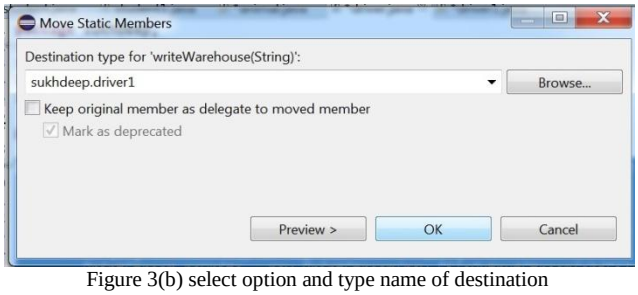

Figure 3(b) select option and type name of destination

See the code that is moved from class driver to driver1 in figure3(c).

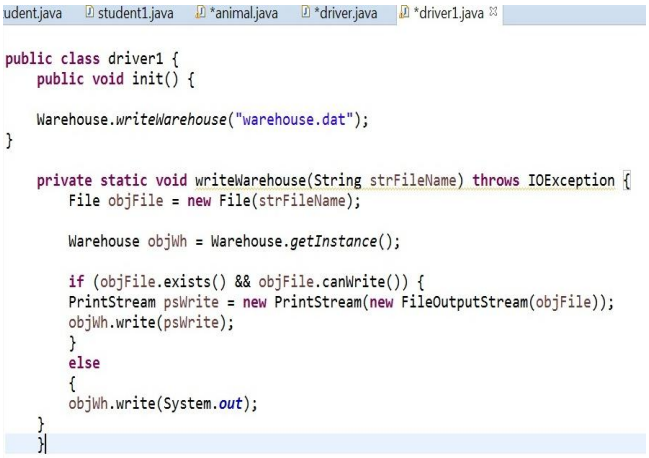

Figure 3(c) code after move refactoring

4. Pull up Method: The Pull up Method is the process of taking a method and "Pulling" it up in the inheritance chain procedure. This is used when a method needs to be used by multiple implementers.

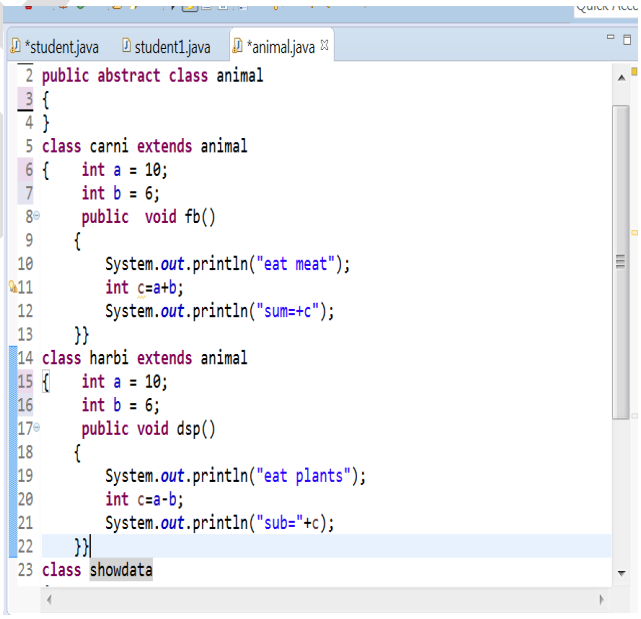

Figure 4(a) code before pull up refactoring

In the figure4 (a) animal is an abstract baseclass, classcarni and harbi derived from it. Both of these classes have same value of variables declare that means duplicate code is available.

For this pull up variables to super class. For this select Refactor >Pull up option and select the members to move for abstract animal baseclass that remove duplicate code shown in the figure4 (b).

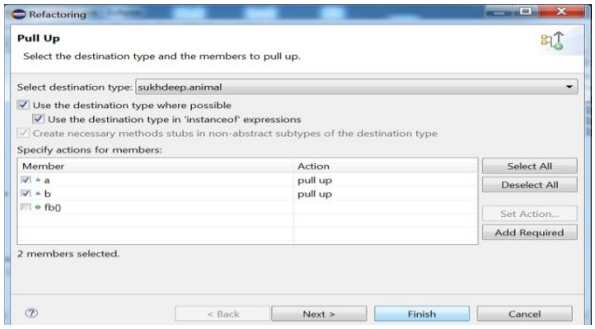

Figure 4(b) select option and members for pull up

The result of code after refactoring in figure 4(c).

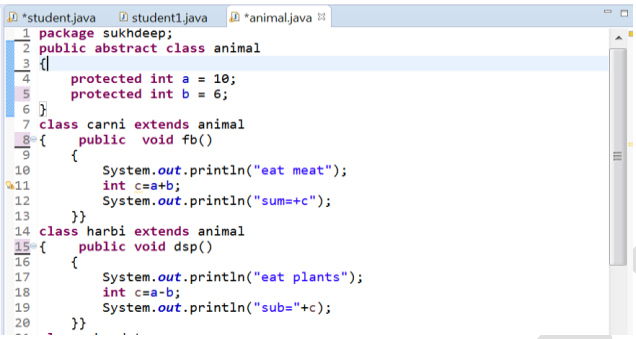

Figure4 (c) code after pull up refactoring

V. RESULTS AND DISCUSSION

Various software refactoring techniques are available that are used to dispose the bad smells. These techniques are implemented using Eclipse tool which support the java language. First, Rename refactoring technique is implemented. Figure1 shows the code before refactoring that does not explain the purpose why to use this class. This code needs to refactor for better understanding and easy to use. The result of rename technique is good as compared to before refactoring. Second, Extract refactoring technique is implemented. Figure2 shows the code before refactoring that does not explain the purpose why to use this code. For effective understanding, extract technique makes new method that gives the meaning.

The result of extract technique reduces the complicated code to simple code but extract code require more space and time for execution. Third, Move technique is implemented. Figure3 shows the code before refactoring that have heavy amount of code but this code mostly used by other class from where it is currently located. This code is move to that class which is more interested. The result of move refactoring technique makes the code more flexible. Fourth, Pull up technique is implemented. Figure4 shows the code before refactoring that use same field in two places. Pull up technique refactor the code from subclasses to superclass. The Pull up technique reduces the duplicate code from program and makes it consistent.

VI. COMPARISON BETWEEN SOME REFACTORING TECHNIQUES

Refactoring Techniques	**Description**	**Use for bad smells**
Rename Method	This is simple and most effective refactoring to rename a property / attribute, method or object and rename identifiers used to reduce the need of comments from a code. Rename method gives more clarity to understand the code. Shortcut: Alt+Shift+R	Code Comments
Extract Method	Extract Method is typically used when a method is too long and used to clean up lengthy, cluttered, complicated code and it also require more space and time. Shortcut: Alt+Shift+M	Long method
Move Method	One class method use another class more than where it is currently located, move to that class. This method used to move static fields, static methods and other elements from one class to another. Shortcut: Alt+Shift+V	Feature Envy
Pull up Method	If two subclasses have the same field then move it to super class. This method used to moves a field or method to super class where it declare as abstract in super class.	Duplicated code

VII. CONCLUSION

Refactoring is an important part of software development cycle and refactoring tools are faster and used to reduce the bug occurrence from the code. In this paper various code smells are defined that are removed by refactoring techniques. These refactoring techniques are implement using refactoring tool that is Eclipse tool and it provide advantages like easy to understand the code, reduce duplicate and inconsistent code. Comparison of refactoring techniques observed that pull up method is better as compared to extract methods because pull up method dispose the complexity and duplicate code from program.

REFERENCES

[1]. M. Fowler, K. Beck "*Refactoring: Improving the Design of Existing Code"* Addison Wesley, 2002

[2]. Alejandra Garrido, Gustavo Rossi, Damiano Distante "*Refactoring for usability in web applications"* IEEE Software Vol.28, No. 3, 2011.

[3]. Mesfin Abebe, Cheol-Jung Yoo " *Trends, Opportunities and Challenges of Software Refactoring: A Systematic Literature Review"* International Journal of Software Engineering and Its Applications Vol.8, No.6, pp.299-318, 2014.

[4]. T. Mens, T. Tourwe. "*A Survey of Software Refactoring*," IEEE Transactions on Software Engineering, Vol. 30, No.2, 2004.

[5]. J. Fields, S. Harvie, M.Fowler, K. Beck; "*Refactoring in Ruby"*, Addison Wesley, 2009

[6]. Mr. Karnam Sreenu, Mr. D. B. Jagannadha Rao "*Detection of Bad Smells In Code for Refactoring Methods"* International Journal of Modern Engineering Research(IJMER)Vol.2, No.5, pp-3727-3729, 2012.

[7]. Francesca Arcelli Fontana, Pietro Braione, Marco Zanoni "*Automatic detection of bad smells in code"* In Journal of Object Technology, vol. 11, no. 2, 2012, pp 1–38, 2012.

[8]. Piyush Chandi "*A survey of code optimization using refactoring"* International Journal on Computer Science and Engineering (IJCSE),Vol. 5 No.4, 2013

[9]. David Gallardo "*Refactroing for everyone*" How and Why use Eclispe's automated refactoring features, IBM.

[10]. http://www.integralist.co.uk/posts/refactoring-techniques.html.