# VIETNAM GENERAL CONFEDERATION OF LABOUR

# TON DUC THANG UNIVERSITY

# FACULTY OF INFORMATION TECHNOLOGY



# DESIGN PATTERN

# MAKING METHODS CALL SIMPLER

# *Supervisor:* MR.NGUYEN THANH PHUOC

# *Author*: NGUYEN MINH NHUT– 518H0545

Class: **18H50203 – 18H50**

Course: **22**

**HO CHI MINH CITY, 2022**

# VIETNAM GENERAL CONFEDERATION OF LABOUR

# TON DUC THANG UNIVERSITY

# FACULTY OF INFORMATION TECHNOLOGY



# DESIGN PATTERN

# MAKING METHODS CALL SIMPLER

# *Supervisor:* MR. NGUYEN THANH PHUOC

# *Author*: NGUYEN MINH NHUT– 518H0545

Class: **18H50203 – 18H50**

Course: **22**

HO CHI MINH CITY, 2022

Appreciation Letter

Firstly, this should be an honor to send my regards to the Faculty of Information Technology, lecturers and staff from all departments of Ton Duc Thang University. I would like to express my sincere thanks for the support and assistance during the implementation of the statistics and probability report.

I would like to express my gratitude to Mr. Nguyen Thanh Phuoc - teachers who directly instructed and supervised me to complete this essay.

I sincerely thank my friends and classmates who are studying and working at Ton Duc Thang University and the family has encouraged, facilitated and helped me during the process.

Due to the fact that my actual ability is still weak, I ensure that I still have many shortcomings, so I hope my supervisor and the other professors will ignore it. At the same time, I hope to receive many comments from many sources to help me accumulate more experience to complete the upcoming graduation report to achieve better results.

THE ESSAY HAS BEEN CONDUCTED IN TON DUC THANG UNIVERSITY

I assure that this is my own product and has been guided by Mr.Nguyen Thanh Phuoc. The research contents, results in this topic are all about honesty. The data in the tables for analysis, comments and evaluation are collected by the me from various sources in the reference section.

In addition, comments and assessments as well as data from other authors or organizations are also used in the essay but with references and annotations.

**If there is any fraud is detected, I ensure my complete responsibility for the contents of my work.** Ton Duc Thang University is not related to violations of authority and copyright caused by me during my work process (if any).

*Ho Chi Minh City, Saturday, 9th April, 2022*

*Authors*

*(Sign and provide full name)*

*Nguyen Minh Nhut*

*Nguyen Khanh Vinh*

VERIFICATION AND EVALUATION FROM LECTURER

**Supervisor’s evaluation**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ho Chi Minh city, date:

(Sign and provide full name)

**Marking lecturer’s evaluation**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ho Chi Minh city, date:

(Sign and provide full name)

ABSTRACT

**Chapter 1:** Rename Method

**Chapter 2:** ToDoNhut’s functions and description

**Chapter 3:** App demo and developer options

**Chapter 4:** References

TABLE OF CONTENTS

[VIETNAM GENERAL CONFEDERATION OF LABOUR 1](#_Toc102023235)

[TON DUC THANG UNIVERSITY 1](#_Toc102023236)

[FACULTY OF INFORMATION TECHNOLOGY 1](#_Toc102023237)

[DESIGN PATTERN 1](#_Toc102023238)

[MAKING METHODS CALL SIMPLER 1](#_Toc102023239)

[Supervisor: MR.NGUYEN THANH PHUOC 1](#_Toc102023240)

[Author: NGUYEN MINH NHUT– 518H0545 1](#_Toc102023241)

[VIETNAM GENERAL CONFEDERATION OF LABOUR i](#_Toc102023242)

[TON DUC THANG UNIVERSITY i](#_Toc102023243)

[FACULTY OF INFORMATION TECHNOLOGY i](#_Toc102023244)

[DESIGN PATTERN i](#_Toc102023245)

[MAKING METHODS CALL SIMPLER i](#_Toc102023246)

[Supervisor: MR. NGUYEN THANH PHUOC i](#_Toc102023247)

[Author: NGUYEN MINH NHUT– 518H0545 i](#_Toc102023248)

[HO CHI MINH CITY, 2022 i](#_Toc102023249)

[Appreciation Letter ii](#_Toc102023250)

[VERIFICATION AND EVALUATION FROM LECTURER v](#_Toc102023251)

[Supervisor’s evaluation v](#_Toc102023252)

[Marking lecturer’s evaluation v](#_Toc102023253)

[ABSTRACT vi](#_Toc102023254)

[TABLE OF CONTENTS 1](#_Toc102023255)

[LIST OFTABLES AND ILLUSTRATIONS 4](#_Toc102023256)

[LIST OF ILLUSTRATION 4](#_Toc102023257)

[CHAPTER 1- RENAME METHODS (FUNCTIONS) 5](#_Toc102023258)

[1.1 Problem: 5](#_Toc102023259)

[1.2 Motivation and Solution: 5](#_Toc102023260)

[1.3 Mechanics – How to rename? 6](#_Toc102023261)

[1.4 Implementation: 6](#_Toc102023262)

[CHAPTER 2 – ADD PARAMETER 8](#_Toc102023263)

[2.1 Problems 8](#_Toc102023264)

[2.2 Motivation and Solution 8](#_Toc102023265)

[2.3 Mechanics 9](#_Toc102023266)

[CHAPTER 3: REMOVE PARAMETER 10](#_Toc102023267)

[3.1 Problems 10](#_Toc102023268)

[3.2 Motivation and Solution 10](#_Toc102023269)

[3.3 Mechanics 11](#_Toc102023270)

[CHAPTER 5: PARAMETERIZE METHOD 12](#_Toc102023271)

[Problems 12](#_Toc102023272)

[Motivation and Solution 12](#_Toc102023273)

[Mechanics 13](#_Toc102023274)

[Implementation 13](#_Toc102023275)

[CHAPTER 6: REPLACE PARAMETER WITH EXPLICIT METHODS 13](#_Toc102023276)

[Problem 13](#_Toc102023277)

[Motivation and Solution 13](#_Toc102023278)

[Mechanics 14](#_Toc102023279)

[Implementation 14](#_Toc102023280)

[CHAPTER 7: PRESERVE WHOLE OBJECT 14](#_Toc102023281)

[Problem 14](#_Toc102023282)

[Motivation and Solution 14](#_Toc102023283)

[Mechanics 15](#_Toc102023284)

[Implementation 15](#_Toc102023285)

[CHAPTER 8: REPLACE PARAMETER WITH METHOD 15](#_Toc102023286)

[Problem 15](#_Toc102023287)

[Motivation and Solution 15](#_Toc102023288)

[Mechanics 16](#_Toc102023289)

[Implementation 16](#_Toc102023290)

[CHAPTER 9: INTRODUCE PARAMETER OBJECT 16](#_Toc102023291)

[Problem 16](#_Toc102023292)

[Motivation and Solution 17](#_Toc102023293)

[Mechanics 17](#_Toc102023294)

[Implementation 17](#_Toc102023295)

[REFERENCES 18](#_Toc102023296)

LIST OFTABLES AND ILLUSTRATIONS

**LIST OF ILLUSTRATION**

[Picture 1.1: Flutter’s log (source) 5](#_Toc58487549)

[Picture 1.2: Flutter codebase example 6](#_Toc58487550)

[Code snippet 1.1: Text sample 7](#_Toc58487551)

[Picture 1.3: Text sample’ result 7](#_Toc58487552)

[Picture 2.1: ToDoNhut’s starting screen 9](#_Toc58487558)

[Picture 2.2: Step 1 to create task 10](#_Toc58487560)

[Picture 2.3: Step 2 to create task 11](#_Toc58487561)

[Picture 2.4: Finish creating task 12](#_Toc58487562)

[Picture 2.5: Task infor screen 13](#_Toc58487564)

[Picture 2.6: Edit Task screen 14](#_Toc58487565)

[Picture 2.7: Mark as complete 15](#_Toc58487567)

[Picture 2.8: Delete a task 16](#_Toc58487569)

[Picture 2.9: Delete a task 17](#_Toc58487571)

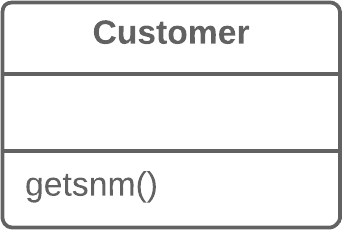
[Picture 2.10: Delete a task 18](#_Toc58487573)

[*Picture 3.1: ToDoNhut Github repo* 19](#_Toc58487576)

# CHAPTER 1- RENAME METHODS (FUNCTIONS)

## Problem:

Imagine that you have a method with a name that you have no idea what it means at the very first time you look at it, for example:

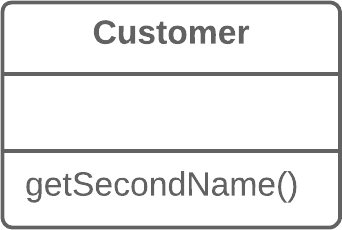


#### Picture 1.1: An example method that you have idea about([source](https://refactoring.guru/images/refactoring/diagrams/Rename%20Method%20-%20Before.png?id=7943798ae9db6b5b232860eed6262462))

As you can see, a method name like this - *getsnm()* will make you confused, me myself came up with *getsupernaturalmother()* and I can assure that you will have many variations popping up in your head so the problem here is that **the name of a method doesn’t clarify its purpose**.

## 1.2 Motivation and Solution:

What we are going to do here to make it less painful every single time we are trying to get a random method’s purpose? Yes, we **shall rename it.**



#### Picture 1.2: Rename that method to getSecondName() so it will be more clear

Sometimes, we can have some sort of offbeat question like: “Why we call a lemon ‘a lemon’?”. It may not make any sense but it is what it is, when you first see thing and people call it “abc”, you have to believe it is “abc” without knowing why it is called like that but it is totally fine to give it a position in your head because everyone declares it like that to make it become a social term

But methods are different, if you keep believing it, you will spend the rest of your life believing *getsnm() is getsupernaturalmother()* but originally, this method gets you the second name of someone. Then we can see one thing, method must be named in the way that communicates their intention. A good wat to do this is to think what the comment for the method would be and turn that comment into the name of the method.

If you ever heard about the term *obfuscation*, you may hate it since it brings you nothing but confusion and it has the same vibe when you are holding a bad-named method so whenever you see it then it is imperative to rename it. You will not get it right at the first time but do not get bored after that because good naming is a skill that requires practice.

## Mechanics – How to rename?

1. See whether the method is defined in a superclass or subclass. If so, you must repeat all steps in these classes too.
2. The next method is important for maintaining the functionality of the program during the refactoring process. Create a new method with a new name. Copy the code of the old method to it. Delete all the code in the old method and, instead of it, insert a call for the new method.
3. Find all references to the old method and replace them with references to the new one.
4. Delete the old method. If the old method is part of a public interface, don’t perform this step. Instead, mark the old method as deprecated.

## Implementation:

For example, we have a method like this:

**public** String **getTelephoneNumber**() {

**return** "(" + officeAreaCode + "-" + officePhoneNumber + ")";

}

This code block will do nothing but get Telephone number of someone but who? The question here is that you need a specific object, a true target. Then we can rename the method

to this:

**public** String **getTelephoneNumber**(){

**return** getOfficeTelephoneNumber;

}

**public** String **getOfficeTelephoneNumber**(){

**return** "(" + officeAreaCode + "-" + officePhoneNumber + ")";

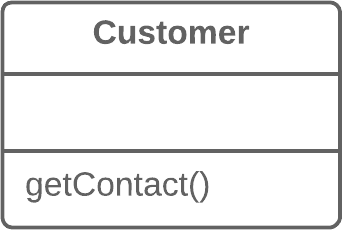
}

Frist, I create a new method and name it **getOfficeTelephoneNumber()**then copy all the things in the body to the new one. When finish, we shall delete the old method and make sure all the callers call the new one.

# CHAPTER 2 – ADD PARAMETER

## 2.1 Problems

More than just its name, a method needs to be clarified with its own parameters where we can say its data must pass on all its functionality. Consider a method below:



#### Picture 2.1: An example method that missing its parameters

## 2.2 Motivation and Solution

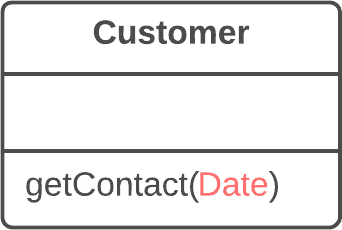
Let say this kind of refactoring is common, you may do that all the time. Everytime you want to change the method and it requires the addition of information so you have to add parameter but adding up things in the program always comes up with alternatives .

But how much missing is missing? Since the act of adding up parameters can lead to longer parameter list but long parameter smells bad because long parameter list is hard to remember and lead to data clumps.

Instead of adding up parameters, we can reference the existing ones, we can ask ourselves these questions:

* Can you ask one of those objects for the information you need?
* If not, would it make sense to give them a method to provide that information?
* What are you using the information for? Should that behavior be on another object, the one that has the information?

But we are talking about adding up parameter so our main solution is still **adding parameter** but think about the alternatives before making your list longer.



#### Picture 2.2: Give our method the parameter it needs

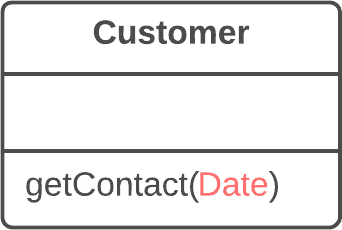
## 2.3 Mechanics

Things are similiar to **Rename method** mechanics

# CHAPTER 3: REMOVE PARAMETER

## 3.1 Problems

More than just its name, a method needs to be clarified with its own parameters where we can say its data must pass on all its functionality. Consider a method below:



#### Picture 2.1: An example method that missing its parameters

## 3.2 Motivation and Solution

Programmers often add parameters but are reluctant to remove them. After all, a spurious parameter doesn't cause any problems, and you might need it again later. This is the demon Obfuscatis speaking; purge him from your soul! A parameter indicates information that is needed; different values make a difference. Your caller has to worry about what values to pass. By not removing the parameter you are making further work for everyone who uses the method. That's not a good trade-off, especially because removing parameters is an easy refactoring.

The case to be wary of here is a polymorphic method. In this case you may well find that other

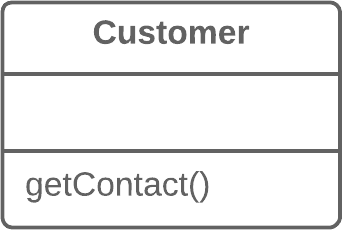
implementations of the method do use the parameter. In this case you shouldn't remove the

parameter. You might choose to add a separate method that can be used in those cases, but you

need to examine how your callers use the method to see whether it is worth doing that. If some

callers already know they are dealing with a certain subclass and doing extra work to find the

parameter or are using knowledge of the class hierarchy to know they can get away with a null, add an extra method without the parameter. If they do not need to know about which class has which method, the callers should be left in blissful ignorance.



#### Picture 2.2: Give our method the parameter it needs

## 3.3 Mechanics

Things are similiar to **Add method** mechanics

# CHAPTER 5: PARAMETERIZE METHOD

## Problems

Multiple methods perform similar actions that only different in their internal values, numbers, or operations.

If you have similar methods, you probably have duplicate code. Or if you need to add yet another version of this functionality, you will have to create yet another method. Instead, you could simply run the existing method with a different parameter.

Diagram

Description automatically generated

Figure 1: An example method of multiple method

## Motivation and Solution

Create a new method, combine these methods by using Parameter that will pass only necessary special values. Or simplify matters by replacing the separate methods with a single method that handles the variations by parameters.

Removes duplicate code and increase flexibility, simple way to deal with other variations by adding paramters.

Table

Description automatically generated

Figure 2: An example solution for parameterize method

## Mechanics

1. Create a parameterized method that can be substituted for each repetitive method. Then compile
2. Replace one old method with call to the new method
3. Repeat for all the methods, testing after each one.

You cannot do this for the whole method, but you can for a fragment of a method. In this case, first extract the fragment into a method, then parameterize it.

## Implementation

To be continued…

# CHAPTER 6: REPLACE PARAMETER WITH EXPLICIT METHODS

## Problem

A method is split into parts, each of which is run depending on the value of a parameter.

Its run different code depending on the values of an enumerated parameter.

Text

Description automatically generated

Figure 3: An example method that split into parts and depending on parameter

## Motivation and Solution

Replace Parameter with Explicit Methods is the reverse of the Parameterize Method. This method discrete values of a parameter. The caller must decide what it wants to do by setting the parameter, determine a valid parameter value, so you might as well provide different methods and avoid the conditional and gain compile time checking. Furthermore, interface also clear.

Replace Parameter with Explicit Methods is to extract the individual parts of the method into their own methods and call them instead of the original method.

## Mechanics

1. Create an explicit method for each value of the parameter.
2. For each leg of the conditional, call the appropriate new method
3. Compile and test after changing each leg.
4. Replace each caller of the conditional method with a call to the appropriate new method.
5. Compile and test
6. When all callers are changed, remove the conditional method.

## Implementation

To be continued….

# CHAPTER 7: PRESERVE WHOLE OBJECT

## Problem

You are getting several values from an object and passing their values as parameters in a method call.

Text, letter

Description automatically generated

Figure 4: An example of passing object values as parameter

## Motivation and Solution

The situation arises when an object passes several data values from a single object as parameters in a method call. The problem is if the called object needs new data values later, you must find and change all the calls to this method. To avoid this, passing in the whole object which the data came.

Preserve Whole Object often makes the code more readable. Cause long parameters lists can be hard to work because both caller and callee must remember which values were they.

But there is downside of this. When you pass in values, you only need one value from the required object, it is better to pass in the value than to pass the whole object.

## Mechanics

1. Create a new parameter for the whole object from which the data comes.
2. Compile and test.
3. Determine which parameters should be obtained from the whole object.
4. Take one parameter and replace references to it within the method body by invoking an appropriate method on the whole object parameter.
5. Delete the parameter.
6. Compile and test.
7. Repeat each parameter that can be got from the whole object

## Implementation

To be continued….

# CHAPTER 8: REPLACE PARAMETER WITH METHOD

## Problem

An object invokes a method, then passes the result as a parameter for another method. The receiver can also invoke this method.

Graphical user interface, text

Description automatically generated

Figure 5: An example of query method passing result as parameter to another method

## Motivation and Solution

A method that gets a value that is passed in as parameter by another method. Long parameter lists are difficult to understand, and we should reduce them.

One way of reducing parameter lists is to look to see whether the receiving method can make the same calculation. If an object is calling a method on itself, and the calculation for the parameter does not reference any of the parameters of the calling method, then parameter can be removed by turning the calculation into its own method.

So instead of passing the value through a parameter, try placing a query call inside the method body

Text

Description automatically generated

Figure 6: A solution for Replace Parameter with Method Call

## Mechanics

1. If necessary, extract the calculation of the parameter into a method
2. Replace references to the parameter in method bodies with references to the method.
3. Compile and test after each replacement
4. Use Remove Parameter on the parameter.

## Implementation

To be continued…

# CHAPTER 9: INTRODUCE PARAMETER OBJECT

## Problem

Methods contain a repeating group of parameters, a group of parameters that naturally go together.

Table

Description automatically generated

Figure 7: An example of group of parameters that go together

## Motivation and Solution

You will often see a particular group of parameters that tend to be passed together. Several methods may use this group, either on one class or in several classes. And solution is to replace these parameters with an object that carries all this data. This refactoring is useful because it reduces the size of the parameter’s lists, and long parameters lists are hard to understand. The defined accessors on the new object also make the code more readable, makes it easier to understand and modify.

Once you have clumped together the parameters, you soon see behavior that you can also move into the new class. By moving this behavior into the new object, you can remove duplicated code.

Table

Description automatically generated

Figure 8: Solution with Introduce Parameter Object

## Mechanics

1. Create a new class to represent the group of parameters you are replacing. Making the class immutable and then Compile.
2. Use Add Parameter for the new data clump. Use a null for this parameter in all the callers
3. For each parameter in the data clump, remove the parameter from the signature. Modify the callers and method body to use the parameter object for that value
4. Compile and test after you remove each parameter
5. When you have removed the parameters, look for behavior that you can move into the parameter object with Move Method

## Implementation

To be continued…

# REFERENCES

1. [Refactoring: Improving the Design of Existing Code](https://www.amazon.com/Refactoring-Improving-Existing-Addison-Wesley-Signature/dp/0134757599)