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**
NGUYEN KHANH VINH– 518H0076
THI NGOC PHIU– 518h0044

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**
NGUYEN KHANH VINH– 518H0076
THI NGOC PHIU– 518h0044

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

Thi Ngoc Phu

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

TABLE OF CONTENTS

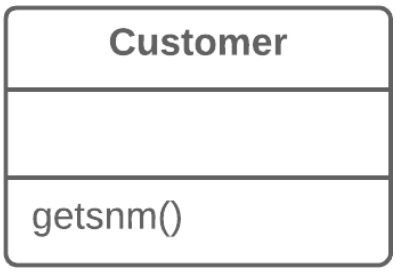
LIST OFTABLES AND ILLUSTRATIONS

LIST OF ILLUSTRATION

CHAPTER 1- RENAME METHODS (FUNCTIONS)

1.1 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:

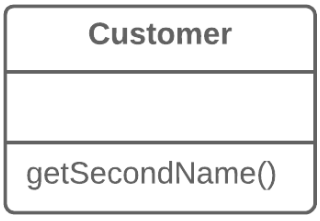


Picture 1.1: An example method that you have idea about(source)

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.

1.3 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.

1.4 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:

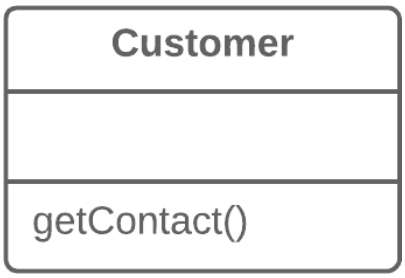
```
public String getTelephoneNumber(){
        return getOfficeTelephoneNumber;
}

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

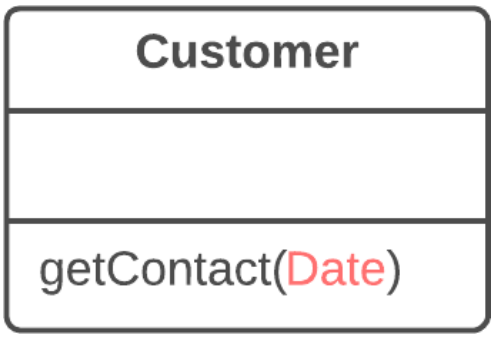
Let say this kind of refactoring is common, you may do that all the time. Every time you want to change the method and it requires the addition of information so you have to add parameter but adding up things in the program always comes up with alternatives.

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

- See whether the method is defined in a superclass or subclass. If the method is present in them, you will need to repeat all the steps in these classes as well.

- The following step is critical for keeping your program functional during the refactoring process. Create a new method by copying the old one and add the necessary parameter to it. Replace the code for the old method with a call to the new method. You can plug in any value to the new parameter (such as null for objects or a zero for numbers).

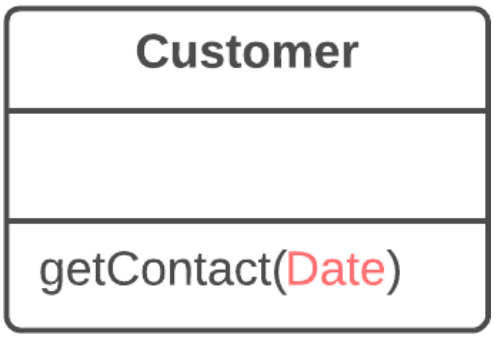
- Find all references to the old method and replace them with references to the new method.

- Delete the old method. Deletion isn't possible if the old method is part of the public interface. If that's the case, mark the old method as deprecated.

CHAPTER 3: REMOVE PARAMETERZ

3.1 Problems

By adding parameters, you will accidentally lengthen the parameter list then you can consider removing the unused parameters. Consider a method below:

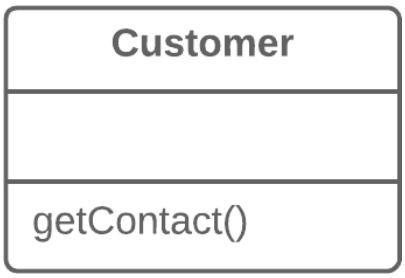


Picture 3.1: An example method that missing its parameters

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 3.2: Give our method the parameter it needs

3.3 Mechanics

- See whether the method is defined in a superclass or subclass. If so, is the parameter used there? If the parameter is used in one of these implementations, hold off on this refactoring technique.

- The next step is important for keeping the program functional during the refactoring process. Create a new method by copying the old one and delete the relevant parameter from it. Replace the code of the old method with a call to the new one.

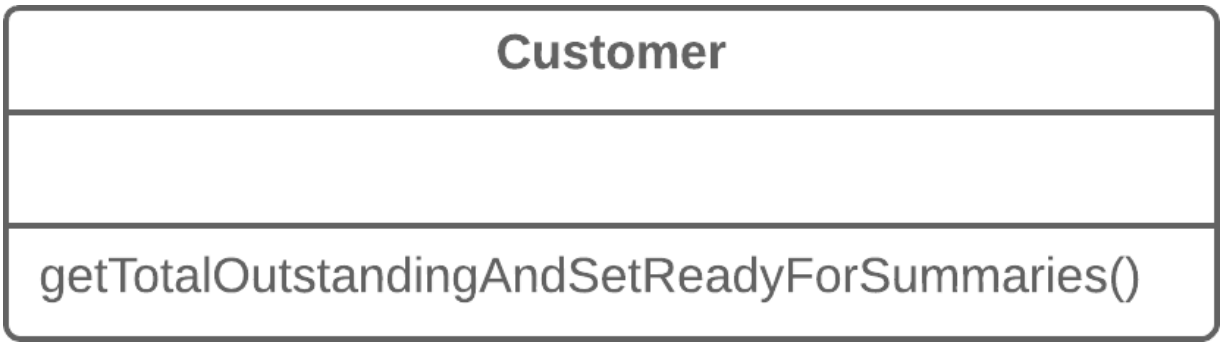
- Find all references to the old method and replace them with references to the new method.

- Delete the old method. Don’t perform this step if the old method is part of a public interface. In this case, mark the old method as deprecated.

CHAPTER 4: SEPARATE QUERY FROM MODIFIER

4.1 Problems

- Believe me or not, one method should handle just one function at a time, for example, **getTelephoneNumber** will just return telephone number of someone but if we have such method that performs 2 more tasks then it is time for us to consider separate the method into something smaller.

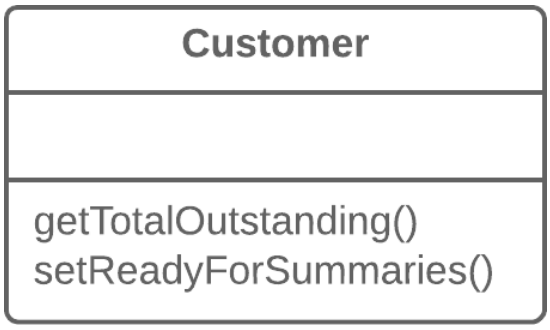


Picture 4.1: An example method that returns a value but also modify something else

4.2 Motivation and Solution

Simply digest this concept by thinking about getter and setter. This solution tends to make amend when you have a method that it performs 2 more tasks at the time such as it shall return a value but also change something inside itself.

By splitting the method into two parts, one for the value and one for the modification



Picture 4.2: Separate the method into getter and setter scenario

4.3 Mechanics

- Create a new *query method* to return what the original method did.
- Change the original method so that it returns only the result of calling the new *query method*.
- Replace all references to the original method with a call to the *query method*. Immediately before this line, place a call to the *modifier method*. This will save you from side effects in case if the original method was used in a condition of a conditional operator or loop.
- Get rid of the value-returning code in the original method, which now has become a proper *modifier method*.

3.4 Implementation

Consider a class below:

```
public class Bill {

        public void sendBill() {
                // Come back later cuz I ran out of energy :(
        }

        public String getTotalOutstandingAndSendBill() {
          String result = "Get total successfully";
          sendBill();
          return result;
        }
```

```
}
```

As we can see, method `getTotalOutstandingAndSendBill` will give you trouble so all the thing you need to do is give it a getter and a setter

```
public class Bill {
        public String totalOutstanding() {
                return "Invoice";
        }

        public void sendBill() {
                // let send some mail functions here
        }
}
```

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.

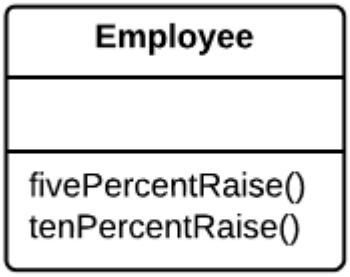


Figure 1: An example method of multiple method

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.

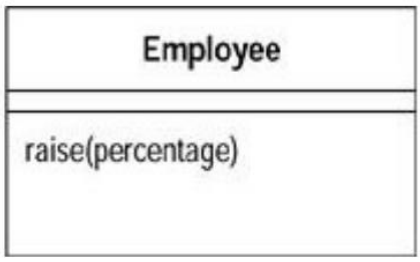


Figure 2: An example solution for parameterize method

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.

```
void setValue (String name, int value) {
    if (name.equals("height"))
        _height = value;
    if (name.equals("width"))
        _width = value;
    Assert.shouldNeverReachHere();
}
```

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.

```
int low = daysTempRange.getLow();
int high = daysTempRange.getHigh();
boolean withinPlan = plan.withinRange(low, high);
```

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.

```
int basePrice = quantity * itemPrice;
double seasonDiscount = this.getSeasonalDiscount();
double fees = this.getFees();
double finalPrice = discountedPrice(basePrice, seasonDiscount, fees);
```

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

```
int basePrice = quantity * itemPrice;
double finalPrice = discountedPrice(basePrice);
```

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.

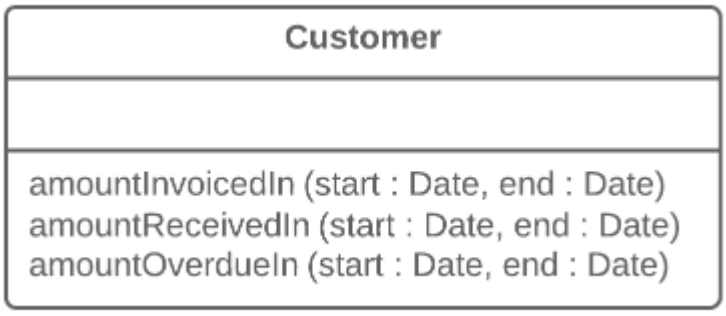


Figure 7: An example of group of parameters that go together

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 assessors 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.

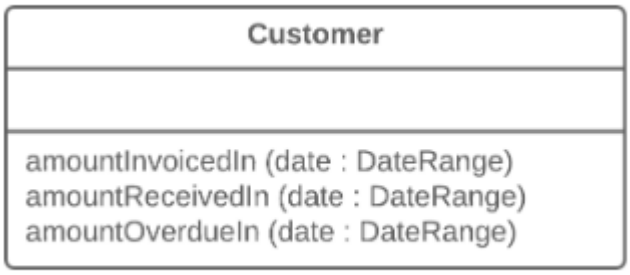


Figure 8: Solution with Introduce Parameter Object

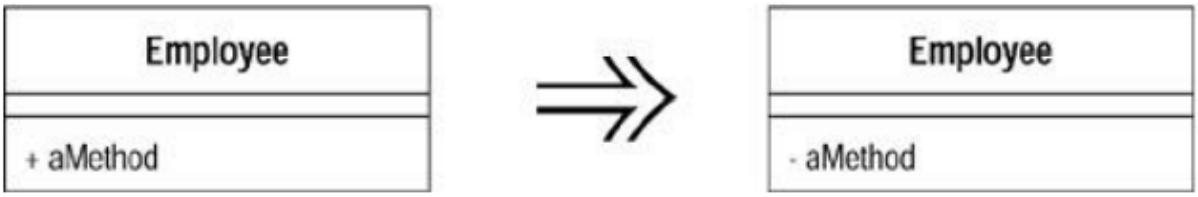
Mechanics

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…

CHAPTER 10: HIDE METHOD



Problem

A method isn't used by other classes or it just be used only inside its own class

Moltivation

If a method is only used inside its own class then it's a waste to leave it public because it may cause some unexpected errors when another class could have access and used it.

One particular case is the getting and setting methods. These two methods are use to interact with the data inside a class, which is belong to that very own class. It may cause the app to break down if another class can manipulate other classes' getter and setter.

Solution:

- Find methods that can be made private
- Make each method as private as possible

CHAPTER 11: REPLACE CONSTRUCTOR WITH FACTORY METHOD

```
Employee (int type) {
    _type = type;
}
```

```
static Employee create(int type) {
    return new Employee(type);
}
```

Problem

A basic setting parameter values in object fields constructor is just not good enough for a complex constructor.

Moltivation

The most obvious motivation for Replace Constructor with Factory Method comes with replacing a type code with subclassing.

- Type code refer to a set of numbers or strings that form a list of allowable values for some entity

After being created with a type code, the object is now in need of creating subclasses. However, subclass is based on the type code and unfortunately, constructor can only return an instance of the object. That's when a static factory method come in handy by returning objects of necessary classes. Moreover, they also have many benefits:

- A factory method can return subclasses based on the arguments given to the method.
- Unlike constructor, they have names that are great in term of distinguish the functionality between classes.
- A factory method can return an already created object.

Solution:

- Create a factory method. Place a call to the current constructor in it.
- Replace all constructor calls with calls to the factory method.
- Declare the constructor private.

1. Replace all constructor calls with calls to the factory method.
2. Declare the constructor private.

CHAPTER 12: ENCAPSULATE DOWNCAST

```
Object lastReading() {
   return readings.lastElement();
}
```

```
Reading lastReading() {
   return (Reading) readings.lastElement();
}
```

Problem

A method returns an object that needs to be downcasted by its callers

Moltivation

- Downcasting is an act of typecasting of a parent object to a child object, which made the parent class inherit the child's properties.

If a value's type returning from a method is known before as more specialized than what the method signature said, the client has to do the downcasting, which is an unnecessary work.

Solution

- Look for cases in which you have to downcast the result from calling a method.
- Move the downcast into the method.

CHAPTER 13: REPLACE ERROR CODE WITH EXCEPTION

```
int withdraw(int amount) {
    if (amount > _balance)
        return -1;
    else {
        _balance -= amount;
        return 0;
    }
}
```

```
void withdraw(int amount) throws BalanceException {
    if (amount > _balance) throw new BalanceException();
    _balance -= amount;
}
```

Problem

Moltivation

Just as our life, code can't run smoothly from the beginning till it ended. It may went south at some point, especially for complicated application where there are hundred of methods being executed simultaneously. It is crucial to know at which part of the code causing the error. Instead of using the good old error code returning conditional statement, Exception in java is the good practice at all. Because it separated the error processing from normal processing, which offer higher readability to the code.

Solution:

- Decide whether the exception should be checked or unchecked
- Find all calls method that return error codes and wrap it in try catch blocks
- Throw an exception inside the method

CHAPTER 14: REPLACE EXCEPTION WITH TEST

```
double getValueForPeriod (int periodNumber) {
    try {
        return _values[periodNumber];
    } catch (ArrayIndexOutOfBoundsException e) {
        return 0;
    }
}
```

```
double getValueForPeriod (int periodNumber) {
    if (periodNumber >= _values.length) return 0;
    return _values[periodNumber];
}
```

Problem

Moltivation

Exception should be use for exceptional behavior, which is an unexpected error. It is such a waste to use exception on conditional checking.

Solution:

- Put a test up front and copy the code from the catch block into the appropriate leg of the if statement.
- Add an assertion to the catch block to notify you whether the catch block is executed.
- Compile and test.
- Remove the catch block and the try block if there are no other catch blocks.
- Compile and test.

REFERENCES

1. Refactoring: Improving the Design of Existing Code
2. Refactor Guru