[Handout for L2P2]
Refactoring: From Turkey to Peacock in Thousand Steps

Refactoring

The first version of the code you write may not be of production quality. In fact, it is acceptable to first concentrate on making the code work rather than worrying too much about the quality of the code, AS LONG AS you improve the quality later. This process of improving a program's internal structure without modifying its external behavior called *refactoring*. Note that,

- Refactoring is not about discarding poorly-written code and rewriting it from scratch. It is the process of improving the code in very small steps until it achieves production quality.
- By definition, refactoring is different from bug fixing or any other modifications that alter the external behavior of the component in concern.

Given below are two common sample refactorings (taken from [1]).

Refactoring name: **Extract method**
Situation: You have a code fragment that can be grouped together.
Method: Turn the fragment into a method whose name explains the purpose of the method.
Example:
//before

```
        void printOwing() {
                printBanner();

                //print details
                System.out.println ("name:        " + _name);
                System.out.println ("amount      " + getOutstanding());
        }
```

//after

```
        void printOwing() {
                printBanner();
                printDetails(getOutstanding());
        }

        void printDetails (double outstanding) {
                System.out.println ("name:        " + _name);
                System.out.println ("amount      " + outstanding);
        }
```

Name: **Consolidate Duplicate Conditional Fragments**
Situation: The same fragment of code is in all branches of a conditional expression.
Method: Move it outside of the expression.
Example:

```
//before
        if (isSpecialDeal()) {
                        total = price * 0.95;
                        send();
                }
                else {
                        total = price * 0.98;
                        send();
                }

//after
        if (isSpecialDeal()){
                        total = price * 0.95;
                }else{
                        total = price * 0.98;
                }
        send();
```

IDEs have built in support for certain refactoring such as automatically renaming a variable/method/class in all places it has been used. Refactoring, even if done with the aid of the IDE, can still result in regressions. Therefore, each small refactoring should be followed by running all the regression tests. Given below are some more commonly used refactorings. A more comprehensive list is available at [1].

1. Consolidate Conditional Expression
2. Decompose Conditional
3. Inline Method
4. Introduce Explaining Variable
5. Remove Double Negative
6. Replace Magic Number with Symbolic Constant
7. Replace Nested Conditional with Guard Clauses
8. Replace Parameter with Explicit Methods
9. Reverse Conditional
10. Split Loop
11. Split Temporary Variable

Whether a refactoring improves the code or degrades it depends on the situations. That is why some refactorings are opposites of each other (e.g., *extract method* vs *inline method*).

It is important to refactor frequently so that 'messiness' in the code does not accumulate and go out of control. It is also important to apply regression testing hand in hand with refactoring to prevent the code from regressing as a result of refactoring.

References

[1] http://refactoring.com/catalog/index.html - This is a list of common refactorings, maintained by Martin Fowler, a leading authority on refactoring. He

is also the author of top book about refactoring: Refactoring: Improving the Design of Existing Code

Worked examples

[Q1]
Do you agree with the following statement? Justify your answer.

Statement: Whenever we refactor code to fix bugs, we need not do regression testing if the bug fix was minor.

[A1]
DISAGREE.

1. Even a minor change can have major repercussions on the system. We MUST do regression testing after each change, no matter how minor it is.
2. Fixing bugs is technically not refactoring.

---End of Document---