Identification of Refactoring Opportunities in Java Programs

With Main Focus on “Replace Conditional with Polymorphism” Refactoring

Refactoring

Refactoring

Program transformation that restructures a program by preserving its behaviour.

Refactoring

Refactoring

Program transformation that restructures a program by preserving its behaviour.

Example

Rename, Move Class, Extract Method, Eliminate Duplicate Code, Replace Conditional with Polymorphism and so on

Refactoring

Refactoring

Program transformation that restructures a program by preserving its behaviour.

Example

Rename, Move Class, Extract Method, Eliminate Duplicate Code, Replace Conditional with Polymorphism and so on

Why to Refactor?

- Improves readability and maintainability of the code
- Helps in finding bugs

Refactoring

What has been done?

Refactoring

What has been done?

- Manual refactoring

  $\Longrightarrow$ tedious and error-prone

Refactoring

What has been done?

- Manual refactoring
  $\Longrightarrow$ tedious and error-prone

- To address the problems in manual refactoring, several semi-automated tools have been developed
  - Smalltalk Refactoring Browser, IntelliJ Idea, Eclipse and so on.
  - User identifies the parts of the software to be refactored which requires a lot of effort and time for large scale systems.

Refactoring

What has been done?

- Manual refactoring
  $\Longrightarrow$ tedious and error-prone

- To address the problems in manual refactoring, several semi-automated tools have been developed
  - Smalltalk Refactoring Browser, IntelliJ Idea, Eclipse and so on.
  - User identifies the parts of the software to be refactored which requires a lot of effort and time for large scale systems.

- To reduce the efforts involved in manually identifying the refactoring opportunities, techniques for automatic identification have been proposed
  - Identification of Move Method refactorings, the recommendation of Rename refactorings and so on

Refactoring (Contd..)

What is yet to be addressed?

- As per our knowledge, automatic identification for many non-trivial refactorings is not yet done.

Refactoring (Contd..)

What is yet to be addressed?

- As per our knowledge, automatic identification for many non-trivial refactorings is not yet done.

Goal

Automate the identification process of refactoring opportunities with static analyses efficiently.

Current Goal

Automatic identification of refactoring opportunities for "Replace Conditional with Polymorphism" refactoring.

Motivating Example

- If statement is used to simulate polymorphism in this example.
- The field base of the class Number is compared against values 2/8/16 to check for a state of the object o1 and call the state specific code printBinary()/printOctal()/printHex() (lines 3-9).
- Similarly, many such conditional (If/switch) statements spans across multiple places in the project increasing the complexity of the code.

```
1  class Number{
2    int base;
3    printBinary(){
4      ...
5    }
6    printOctal(){
7      ...
8    }
9    printHex(){
10     ...
11   }    ...  }
```

```
1  class SomeClass{
2   m(Number o1){
3    if(o1.base == 2){
4     o1.printBinary();
5    }else if(o1.base == 8){
6     o1.printOctal();
7    }else if(o1.base == 16){
8     o1.printHex();
9    }
10   ...
11  }  ... }
```

Solution: Replace Conditional with Polymorphism (RCP)

```
class Number{
  print(){
    ...
  }
    ...
}
```

```
class Binary extends Number{
  print(){
    ...
  }
//other methods specific to Binary
}
```

```
class Hex extends Number{
  print(){
    ...
  }
//other methods specific to Hex
}
```

```
class Octal extends Number{
  print(){
    ...
  }
//other methods specific to Octal
}
```

Solution: Replace Conditional with Polymorphism (RCP)

```
class Number{
  print(){
    ...
  }
    ...
}
```

```
class Binary extends Number{
  print(){
    ...
  }
//other methods specific to Binary
}
```

```
class Hex extends Number{
  print(){
    ...
  }
//other methods specific to Hex
}
```

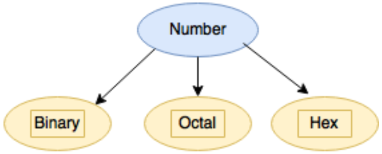
```
class Octal extends Number{
  print(){
    ...
  }
//other methods specific to Octal
}
```

```
class SomeClass{
  m(Number o1){
o1.print();
  }
  ...
}
```

- Creates one subclass per value (2/8/16): Binary, Octal, Hex. i
- Moves the state specific code as a method to the respective sub class.
- Replaces the conditional (If statement) by a polymorphic function call.

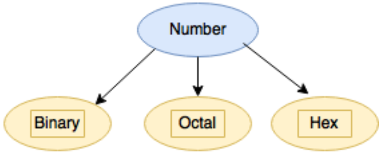
Inheritance Hierarchy

- If the field `base` is not redefined after the object `o1` is created. That is `base` is immutable for `o1`. Then the inheritance hierarchy after the refactoring (*Replace Type Code with Subclass*):

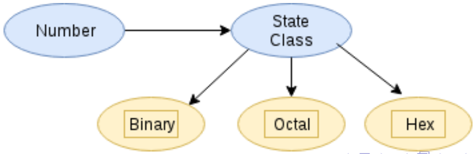


Inheritance Hierarchy

- If the field base is not redefined after the object o1 is created. That is base is immutable for o1. Then the inheritance hierarchy after the refactoring (*Replace Type Code with Subclass*):



- If the field base is redefined after the object o1 is created. Then the inheritance hierarchy after the refactoring (*Replace Type Code with State*):



More Information at

- Replace Conditional with Polymorphism
  - https://www.youtube.com/watch?v=NCsoEEz_Ta0
  - https://refactoring.com/catalog/replaceConditionalWithPolymorphism.html
- Replace Type Code with State
  - https://refactoring.com/catalog/replaceTypeCodeWithStateStrategy.html
  - https://sourcemaking.com/refactoring/replace-type-code-with-state-strategy
- Replace Type Code with Subclass
  - https://refactoring.com/catalog/replaceTypeCodeWithSubclasses.html
  - https://sourcemaking.com/refactoring/replace-type-code-with-subclasses

More Information at

- Replace Conditional with Polymorphism
    - https://www.youtube.com/watch?v=NCsoEEz_Ta0
    - https://refactoring.com/catalog/replaceConditionalWithPolymorphism.html
- Replace Type Code with State
    - https://refactoring.com/catalog/replaceTypeCodeWithStateStrategy.html
    - https://sourcemaking.com/refactoring/replace-type-code-with-state-strategy
- Replace Type Code with Subclass
    - https://refactoring.com/catalog/replaceTypeCodeWithSubclasses.html
    - https://sourcemaking.com/refactoring/replace-type-code-with-subclasses

Thank You!!