Yu-Hong Jhuo (Marcus)

yuj0072@arastudent.ac.nz

BCPR301   
Advanced Programming

Assessment 2 Marking Sheet for Coder

Contents

[Repository link: 2](#_Toc20178398)

[Feature list 2](#_Toc20178399)

[Interface diagram 2](#_Toc20178400)

[Bad smells before refactoring 3](#_Toc20178401)

[Refactoring 1 4](#_Toc20178402)

[Name 4](#_Toc20178403)

[Location 4](#_Toc20178404)

[Reasons 4](#_Toc20178405)

[Strategies/ approaches 5](#_Toc20178406)

[Result Evaluation 5](#_Toc20178407)

[Has the bad smell been removed? 5](#_Toc20178408)

[Did you bring new bad smells into the program? 5](#_Toc20178409)

[How well is your program now in terms of software quality? 5](#_Toc20178410)

[Worst bad smells after refactoring1 5](#_Toc20178411)

[Refactoring 2 5](#_Toc20178412)

[Name 5](#_Toc20178413)

[Location 5](#_Toc20178414)

[Reasons 5](#_Toc20178415)

[Strategies/ approaches 5](#_Toc20178416)

[Result Evaluation 5](#_Toc20178417)

[Has the bad smell been removed? 5](#_Toc20178418)

[Did you bring new bad smells into the program? 5](#_Toc20178419)

[How well is your program now in terms of software quality? 6](#_Toc20178420)

[Worst bad smells after refactoring2 6](#_Toc20178421)

[Refactoring 3 6](#_Toc20178422)

[Name 6](#_Toc20178423)

[Location 6](#_Toc20178424)

[Reasons 6](#_Toc20178425)

[Strategies/ approaches 6](#_Toc20178426)

[Result Evaluation 6](#_Toc20178427)

[Has the bad smell been removed? 6](#_Toc20178428)

[Did you bring new bad smells into the program? 6](#_Toc20178429)

[How well is your program now in terms of software quality? 6](#_Toc20178430)

[Expected marks 6](#_Toc20178431)

# Repository link:

Refactored code repository

<https://github.com/forestraindrip/PR301_Code_Refactoring.git>

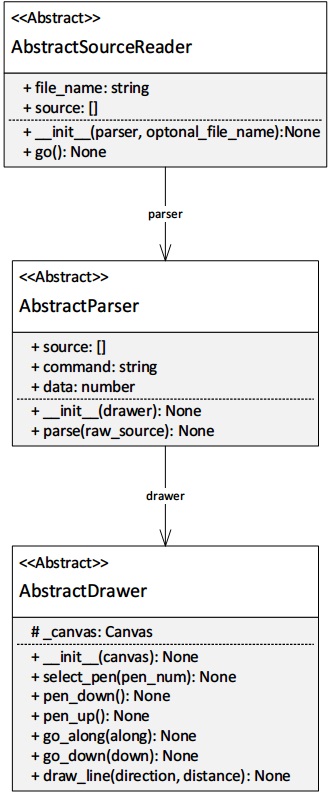
Source code repository

<https://github.com/forestraindrip/PR301_Assignment2.git>

# Feature list

* User can select pen
* User can put pen down
* User can move pen up
* User can move the pen horizontally (go along)
* User can move the pen vertically (go down)
* User can draw a line with input direction and distance
* The system can read commands from text file
* The system can parse the commands from text file

# Interface diagram



# Bad smells before refactoring

The order of the bad smells is listed from the worst bad smell to the lesser ones.

* Lazy Class in AbstractSourceReader

Location: tigr.py => AbstractSourceReader => line 51~64

* Inappropriate Intimacy between frontends and parsers

Location: front\_end\_kieran.py => TkinterInterface => draw() => line 71

front\_end\_jerry.py => GuiInterface => draw() => line 64

* Shotgun Surgery in the drawers and frontends

Location: drawer\_jack.py => line 7~13, 66

drawer\_kieran.py => line 11~17, 55

drawer\_turtle\_jack.py => line 8~14, 19

all frontend classes

* Alternative Classes with Different Interfaces in two frontends.

Location: front\_end\_kieran.py => TkinterInterface

front\_end\_jerry.py => GuiInterface

* Switch statement in the drawers and the parsers.
* Refused-bequest in MainTIGr.
* Long methods in frontends

# Refactoring 1

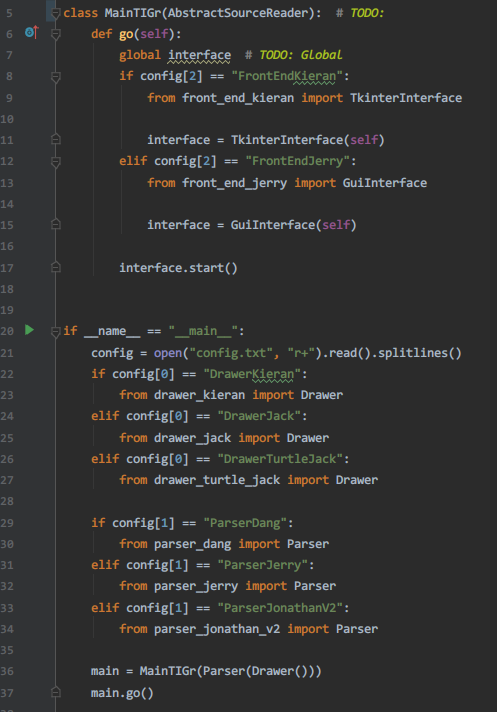
## Name

Lazy Class

## Location

* refactored\_code
  + source\_reader\_kieran.py
    - Whole MainTIGr class
* tigr.py
  + AbstractSourceReader line 48~62

## Reasons



The AbstractSourceReader is the lazy class. As shown in the screenshot, the MainTIGr inherits the AbstractSourceReader. However, there should not be a relationship between them. In assignment 1, the responsibilities of provided abstract class, AbstractSourceReader, is reading source file then passing the result to the parser for further processing; its implementation is compulsory. However, in this case, MainTIGr is used as the entry point of the program which distorts the role of AbstractSourceReader. As a matter of fact, AbstractSourceReader has no actual functionality in the program which essentially makes it a **Lazy Class**. Moreover, the bad implementation **creates other bad smells including** **inappropriate intimacy, shotgun surgery and refused-bequest**. Therefore, I think the Lazy Class of AbstractSourceReader is the most critical bad smell at this stage.

## Strategies/ approaches

* Break the relationship between MainTIGr and AbstractSourceReader.
* Implement a proper SourceReader for source reading.
* Rename the drawers. Three drawers with different implementation sharing the same name before refactoring.
* Rename the parsers. Three parsers with different implementation sharing the same name before refactoring.
* Redirect the functions using source reading back to the created SourceReader.

## Result Evaluation

### Has the bad smell been removed?

Yes

### Did you bring new bad smells into the program?

No, only a new SourceReader is created. It is a well-behaved source reader absented from the original system.

### How well is your program now in terms of software quality?

* Low coupling: The classes, including drawers, source reader and GUI, have lower coupling now.
* No global variable: The refactoring removes the global variable “interface” in the MainTigr class.
* Separated responsibilities of classes: The frontend classes don’t have the responsibility of SourceReader now. The SourceReader is not used as entry point as well.

## Worst bad smells after refactoring1

* Alternative Classes with Different Interfaces in two frontends
* Duplicate code in frontends, drawers
* Long methods in frontends
* Switch statement in the drawers and the parsers

# 

# Refactoring 2

## Name

Alternative Classes with Different Interfaces

## Location

* refactored\_code
  + front\_end\_jerry.py
    - GuiInterface whole class
  + front\_end\_kieran.py
    - TkinterInterface whole class

## Reasons

These two frontend classes have similar functions with different implementations and names. Parts of them have duplicate codes which decreases code readability.

## Strategies/ approaches

1. Extract Superclass: Create a superclass AbstractFrontEnd which is inherited by two frontend classes

## Result Evaluation

### Has the bad smell been removed?

Yes

### Did you bring new bad smells into the program?

No.

### How well is your program now in terms of software quality?

* Code consistency is increased.
* Code readability is increased.
* Code duplication is greatly reduced.

## Worst bad smells after refactoring2

* Long methods in frontends
* Switch statements in parsers and drawers

# 

# Refactoring 3

## Name

Long methods

## Location

* refactored\_code
  + front\_end\_jerry.py
    - GuiInterface
      * init\_widgets() line 19~79

## Reasons

The method has 60 lines. This make the code hard to read and maintain.

## Strategies/ approaches

* Extract Method: Migrate codes in the init\_widgets() into multiple methods and give these methods meaningful names.
* Extract Method: Merge identical method to reduce code duplication after the previous step.

## Result Evaluation

### Has the bad smell been removed?

Yes

### Did you bring new bad smells into the program?

No

### How well is your program now in terms of software quality?

* Code readability is increased.
* Code duplication is greatly reduced.

# 

# Expected marks

1. Smell detection

* Identification of bad smells: 3\*1 marks
* Location of bad smells: 3\*1 marks
* The reasons: 3\*1 marks
* Descriptions of strategies: 3\*1 marks

Total: 12 marks

1. Refactoring
   * Identification of the worst smell: 3\*1 marks
   * Version control: 3\*1 marks
   * Modification and validation: 3\*2 marks
   * Testing and evaluations: 3\*1 marks

Total: 15 marks

Overall: **27 Marks**