## Creator

Yu-Hong, Jhuo

## Worst bad smells before refactoring

* Lazy Class in AbstractSourceReader.
* Inappropriate Intimacy between frontends and parsers.
* Shotgun Surgery in the drawers and frontends.
* Alternative Classes with Different Interfaces in two frontends.
* Switch statement in the drawers and the parsers.
* Refused-bequest in MainTIGr.
* Long methods in frontends

## Refactoring 1

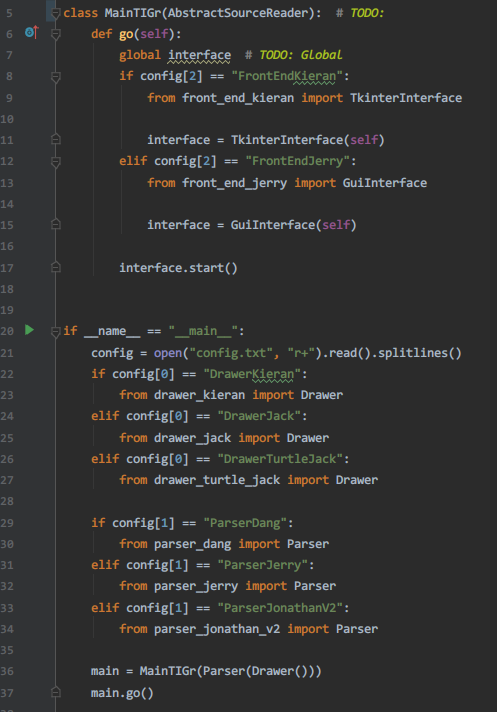
### Name

### Lazy Class

### Location

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

### Reasons



The MainTIGr inherits inappropriate parent class and twist the responsibility of the function in the class. As shown in the screenshot, the MainTIGr inherits the AbstractSourceReader. In assignment 1, the responsibilities of provided abstract class AbstractSourceReader are reading source and handing over the result to the parser for further processing. However, MainTIGr is used as the entry point of the system which distorts the role of AbstractSourceReader. The inheritance is misused in this case. If I break the relationship between AbstractSourceReader and MainTIGr, the AbstractSourceReader becomes a **Lazy Class**. Moreover, the current system distributes the function of reading file into multiples frontend interfaces which **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 parser with different implementation sharing the same name before refactoring.
* Redirect the functions needing source reading to the new 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. In addition, the SourceReader is not used as entry point anymore.

## Worst bad smells after refactoring1

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

## 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.

### Strategies/ approaches

1. Extract Superclass: Create a superclass AbstractFrontEnd which inherited by two 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 pieces of code into several methods and give these methods meaningful names.

### 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.