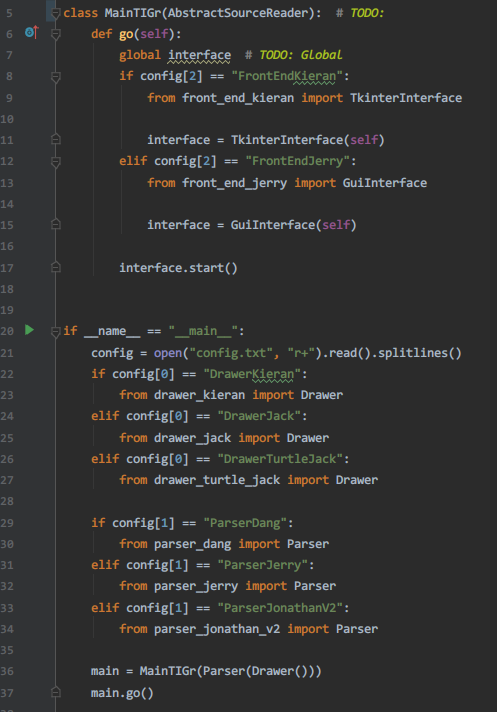
## Major non-functional defect in the program



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 responsibiliteis of provided abstract class AbstractSourceReader are reading source and handing over the result to its parser for further process. However, MainTIGr is used as the entry point of the system. The inheritance is misused in this case.

## 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 Frontend
* Switch statement in the drawers and the parsers
* Refused-bequest in MainTIGr

## Refactoring 1

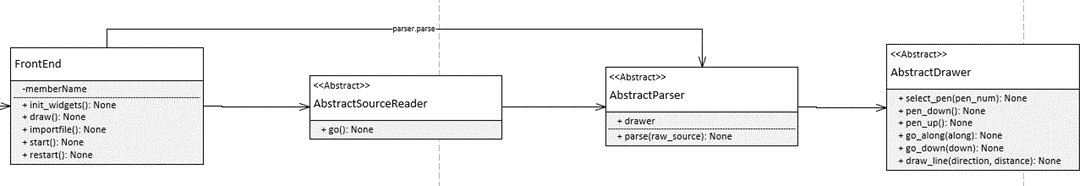
### Name

### Lazy Class

### Location

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

### Reasons



After examination, the current system has no proper implementation of AbstractSourceReader for file reading which is **required for Assignment1**. The only implementation AbstractSourceReader is MainTIGr class and it is used as the entry point of the system; the MainTIGr twists the role of AbstractSourceReader and makes AbstractSourceReader more like a **Lazy Class**. In addition, the current system distributes the function of reading file into multiples front end interface which **create 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.
* Rename the parsers: Three parser with different implementation sharing the same name.
* Redirect the functions need 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 connections now.
* No global variable: The refactoring removes the global variable “interface” in the frontend classes.
* Separated responsibilities of classes: The front-end 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 Frontend
* Switch statement in the drawers and the parsers
* Duplicate code

## 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 classes have similar functions with different implementations and names. Parts of them have duplicate codes

### Strategies/ approaches

1. Extract Superclass: Create a superclass 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

## Refactoring

### Name

Inappropriate Intimacy

### Location

### Reasons

placeholder

placeholder

### Strategies/ approaches