**BCPR301 – Advanced Programming**

**Assessment 3 Marking Sheet**

Student Name: Sini Gao

# The compulsory (i.e., ZERO mark if not being provided)

1. You MUST submit a filled self-marking sheet to indicate how many marks you think you can get for each section in the marking guide provided below.

I think should have 39 marks

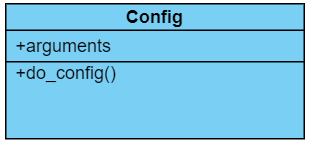
# Marking guide (20 \* N marks in total where N = 2)

1. The class diagram **before** your modification (4 marks)
2. **Strategy Pattern**

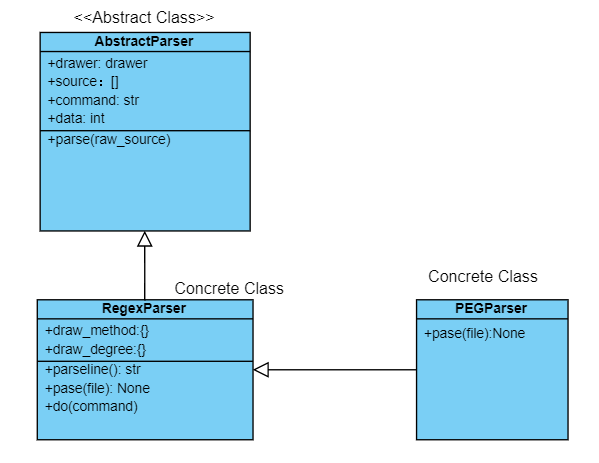
the before code wasn’t in a class, it’s a long method under if \_\_name\_\_ == “\_\_main\_\_” which used for identify and load config files



Before refactoring, I put this long algorithm in a class named Config which only has one method do\_config:



1. **Template Method**



1. The locations of code involved (i.e., code reference) **in your target assignment 1 or 2 solution** (2 \* N marks)

I used my Assignment 1 as a code base for assignment 3’s further improvement.

1. **Strategy Pattern**

Abstract Strategy: tigr.py/line 74 - 78

Concrete Strategy A: tigr.py/line 82 - 89

Concrete Strategy B: tigr.py/line 93 – 104

Client: tigr.py/line 32 - 71

1. **Template Method**

Common interface: tigr\_interface.py/line 36-45

Template Class: parser/template\_parser.py/line 6-79

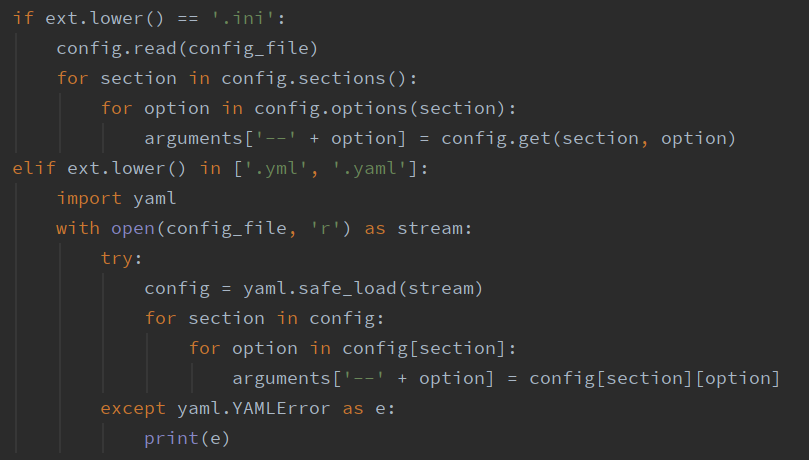
Concrete Class A parser.regex\_parser.py/line 4-15

Concrete Class B peg\_parser.py/line10-44

1. The name of the design pattern applied (4 marks)
2. **Strategy Pattern**
3. **Template Method**
4. The reasons why applying this design pattern is suitable; the reasons **MUST be specified** for the particular situation you try to apply, i.e., **do not just give general reasons** why using that design pattern is good. (3 marks)
5. **Strategy Pattern**

The before code was a long if else statement chain and wasn’t in class, which would be difficult for code reusing and adding new features and against principle of single responsibility

Assignment code can implement 2 type of configure files which are ini and yaml:

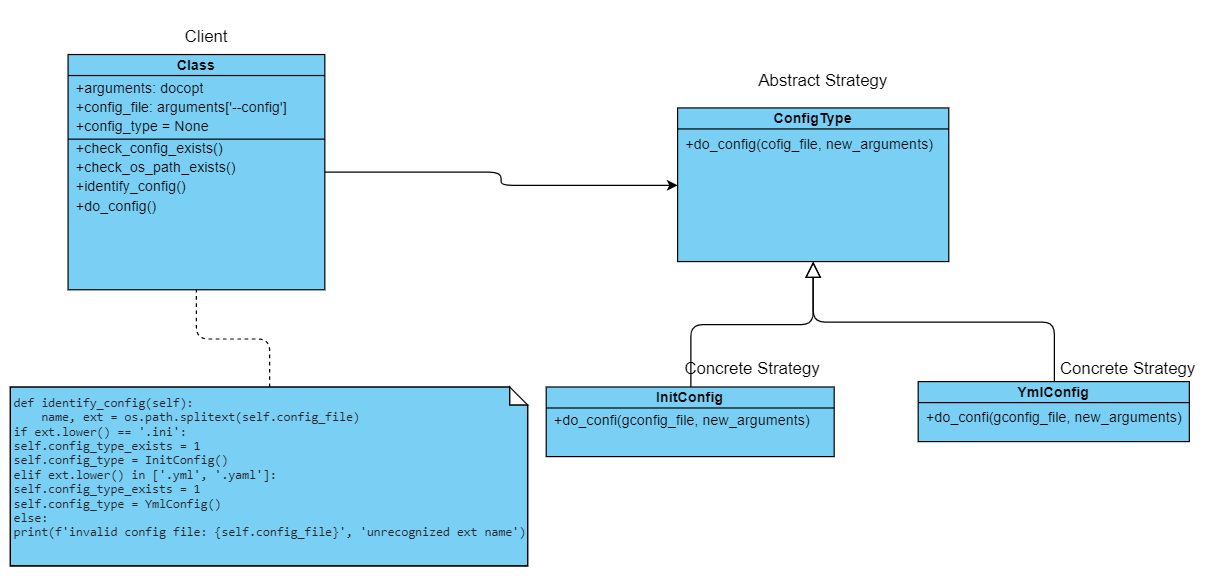


In the future, we may need to implement more types of configure file, applying strategy pattern helps to encapsulate algorithm of loading different configure files and open for code extensions.

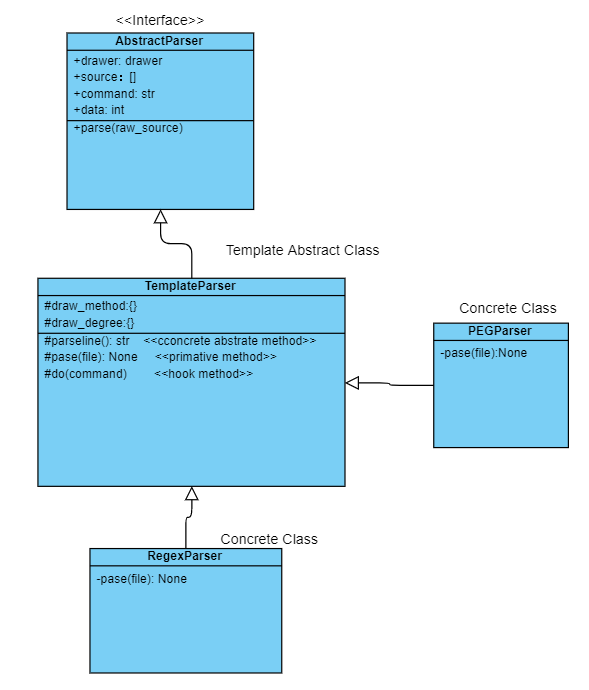
1. **Template Method**

Before applying template method, class PEGParser was a subclass of RegexParser, which result in tight coupling between this two class, it would be hard to modified RegexParser without affecting on PEGParser. Applying Template Method resolve this problem, by creating common superclass for concrete classes to depend on. And later we can extending functionality by subclass an other concrete class of the abstract class.

1. The class diagram after your modification; all the components in the design pattern class diagram provided in our textbook should be **explicitly labelled** in your class diagram. (2 \* N marks)
2. **Strategy Pattern**



1. **Template Pattern**



1. Applying the design pattern proposed. Your assignment 3 solution needs to pass PEP8 check (10 \* N marks)

The assignment 3 solution passed PEP8 check attached code for your reference