Lecture 13: Template Pattern
IN710: Object-Oriented Systems Development
Semester One, 2020

Kaiako: Grayson Orr

Te Kura Matatini ki Otago, Ōtepoti, Aotearoa

Tuesday, 21 April

Lecture 12: Flyweight Pattern Recap

- Design pattern 08: flyweight pattern
  - Definition
  - Problem/solution
  - UML
  - Immutability
  - Implementation
  - Pros & cons

Lecture 13: Template Pattern Topics

- Design pattern 09: template pattern
  - Definition
  - Problem/solution
  - Real world analogy
  - UML & implementation
  - Pros & cons

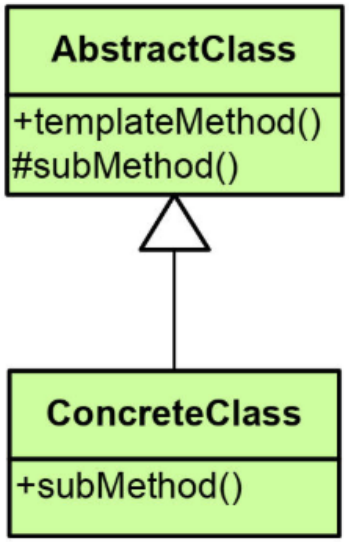
Template Pattern: GoF

- GoF definition & UML

Template Method

Type: Behavioral

What it is:
Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.



Template Pattern: Definition

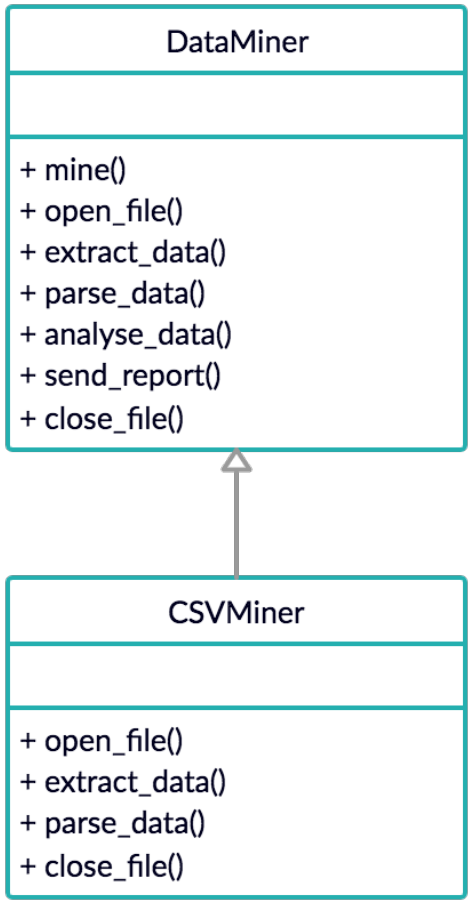
- Behaviourial pattern
- A method in a superclass/abstract superclass
- Defines the skeleton of an operation in terms of a number of high-level steps
- Implemented by helper methods in the same class as the template method
- Helper methods may be either abstract or hook methods
- Intent is to define an overall structure of the operation
- Inversion of control
  - High-level code no longer determines what algorithm to run
  - Instead, a lower-level algorithm is selected at runtime

Template Pattern: Problem

- Data mining application
- Analyses documents in various formats - .pdf, .docx & .csv

Template Pattern: Solution

- DataMiner abstract class
- CSVDataMiner subclass

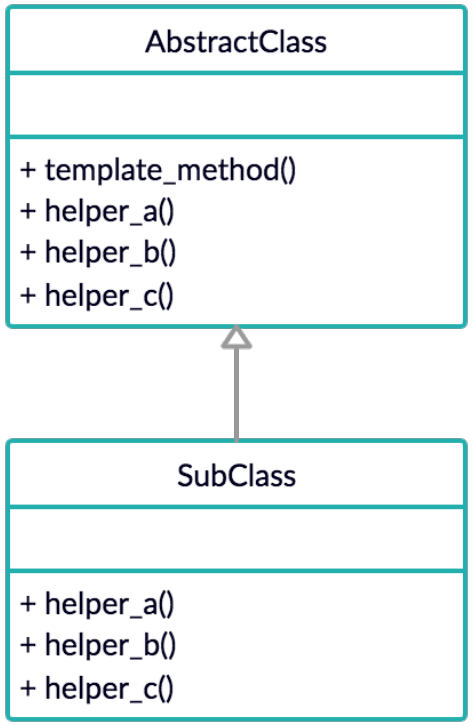


Template Pattern: Real World Analogy

- Building a house
- Each building step can be changed to make the resulting house different from others

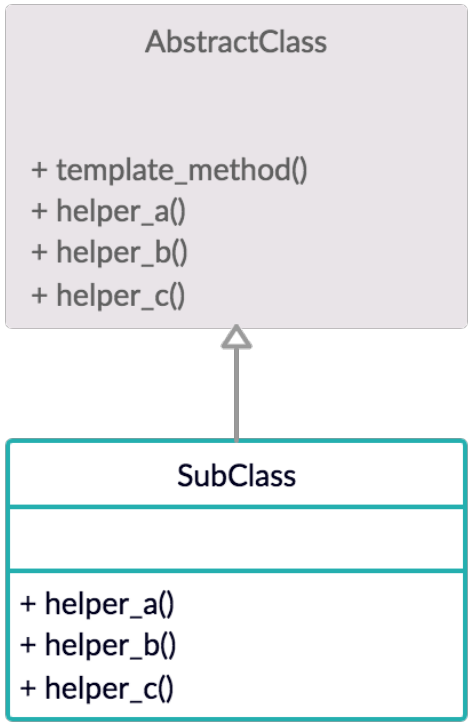
Template Pattern: UML

- Consider the following UML diagram:



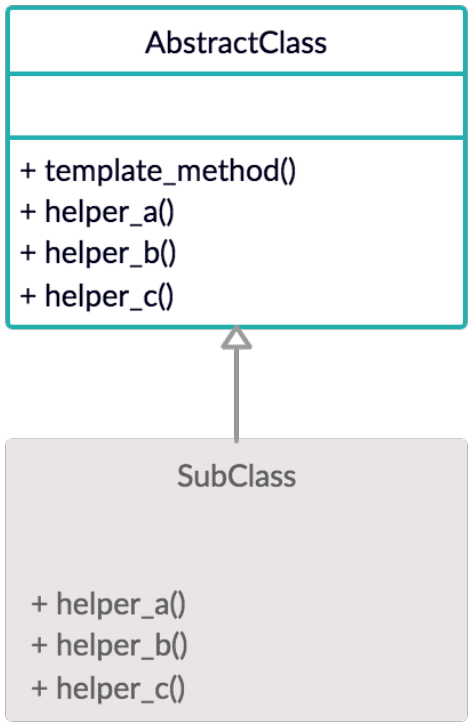
Template Pattern: UML

- Abstract class defines an operation that defines the skeleton/template of a behaviour



Template Pattern: UML

- Allows the subclass to provide its own implementation of the algorithm



Template Pattern: Implementation

```
from abc import ABC, abstractmethod

class MakeDinner(ABC):
    @abstractmethod
    def prepare(self):
        pass

    @abstractmethod
    def cook(self):
        pass

    @abstractmethod
    def eat(self):
        pass

    def cleanup(self):
        pass

    def process(self):
        self.prepare()
        self.cook()
        self.eat()
        self.cleanup()

class MakePizza(MakeDinner):
    def prepare(self):
        print('Preparing the pizza')

    def cook(self):
        print('Cooking the pizza')

    def eat(self):
        print('Eating the pizza')
```

Template Pattern: Implementation

```
class MakeSteak(MakeDinner):
    def prepare(self):
        print('Preparing the steak')

    def cook(self):
        print('Cooking the steak')

    def eat(self):
        print('Eating the steak')

    def cleanup(self):
        print('Cleaning up')

def main():
    make_pizza = MakePizza()
    make_pizza.process()
    make_steak = MakeSteak()
    make_steak.process()

if __name__ == '__main__':
    main()  # Preparing the pizza
            # Cooking the pizza
            # Eating the pizza
            # Preparing the steak
            # Cooking the steak
            # Eating the steak
            # Cleaning up
```

Template Pattern: Pros

- The clients can override only certain parts of a large algorithm
- Duplicate code can be moved into the superclass

Template Pattern: Cons

- There may be limitation to the clients by the provided skeleton/template of an algorithm
- May violate the Liskov Substitution Principle - suppressing a default step implementation
- Harder to maintain the more steps a template method has

Practical

- Series of tasks covering today's lecture
- Worth 1% of your final mark for the Object-Oriented Systems Development course
- Deadline: Friday, 12 June at 5pm

Reminder: Exam 03

- Series of tasks covering lectures 09-12
- Worth 6% of your final mark for the Object-Oriented Systems Development course
- Deadline: Thursday, 23 April at 5pm

Lecture 14: Prototype Pattern Topics

- Design pattern 10: prototype pattern
  - Definition
  - Problem/solution
  - UML & implementation
  - Pros & cons