Lecture 12: Flyweight Pattern
IN710: Object-Oriented Systems Development
Semester One, 2020

Kaiako: Grayson Orr

Te Kura Matatini ki Otago, Ōtepoti, Aotearoa

Thursday, 26 March

Lecture 11: State Pattern Recap

- Design pattern 07: state pattern
  - Definition
  - Problem/solution
  - Real world analogy
  - UML & implementation
  - Pros & cons

Lecture 12: Flyweight Pattern Topics

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

Flyweight Pattern: Definition

- Structural pattern
- An object that minimises memory usage
- Share as much data as possible with other similar objects
- A way to use objects in large numbers when a repeated representation is using an unacceptable amount of memory

Flyweight Pattern: Problem

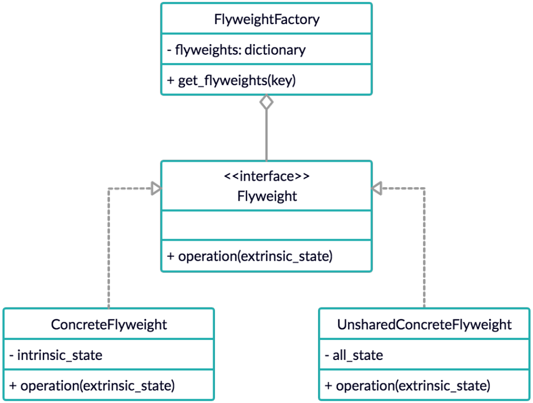
- Word processor application
- Create an object for each character typed
- Contain information such as font face, font size, etc
- A document might contain tens of thousands of characters

Flyweight Pattern: Solution

- Create an object that stores such information
- Example: a document with 750 characters in arial font
- The characters would contain a reference to a flyweight object that stores common information
- The information is only stored once, minimising memory usage

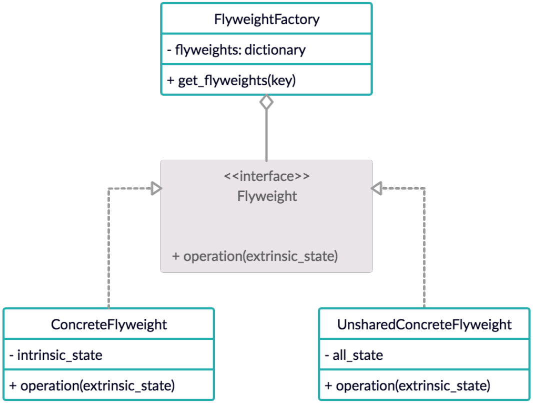
Flyweight Pattern: UML

- Consider the following UML diagram:



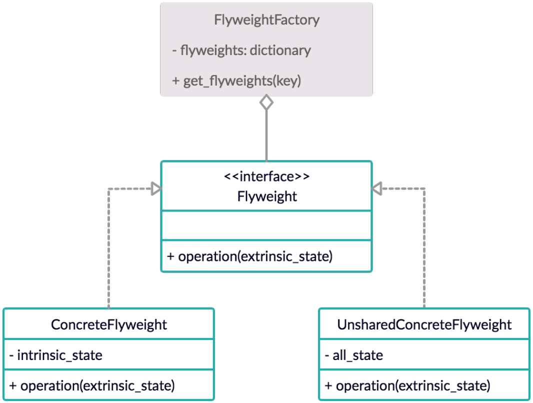
Flyweight Pattern: UML

- Flyweight interface class
- Performs an operation by passing in an extrinsic state



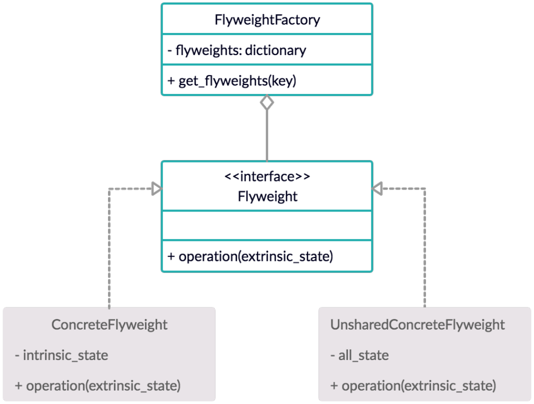
Flyweight Pattern: UML

- Flyweight factory class
- The client refers to the flyweight factory to create/share flyweight objects



Flyweight Pattern: UML

- The concrete flyweight class implements the flyweight interface class
- Stores an intrinsic state that can be shared



Flyweight Pattern: Immutability

- To enable safe sharing between the clients & threads, the flyweight objects must be immutable

Flyweight Pattern: Implementation

```
from json import dumps

class Flyweight:
    def __init__(self, shared_state):
        self.__shared_state = shared_state

    def operation(self, extrinsic_state):
        shared = dumps(self.__shared_state)
        extrinsic = dumps(extrinsic_state)
        print(f'Shared␣state␣-␣{shared}')
        print(f'Extrinsic␣state␣-␣{extrinsic}')

class FlyweightFactory:
    def __init__(self, flyweights):
        self.__flyweights = {}

        for f in flyweights:
            self.__flyweights[self.get_key(f)] = Flyweight(f)

    def get_key(self, state):
        return '␣'.join(state)

    def get_flyweight(self, shared_state):
        key = self.get_key(shared_state)
        if not self.__flyweights.get(key):
            print('Creating␣new␣flyweight')
            self.__flyweights[key] = Flyweight(shared_state)
        else:
            print('Reusing␣existing␣flyweight')
        return self.__flyweights[key]

    def display_flyweights(self):
        flightweight_keys = self.__flyweights.keys()
        flyweights = [f for f in flightweight_keys]
        print(f'List␣of␣flyweights␣-␣{flyweights}')
```

Flyweight Pattern: Implementation

```
def main():
    cars = [
        ('Ferrari', 'SF90_Stradale'),
        ('McLaren', 'Speedtail'),
        ('SSC', 'Tuatara')
    ]
    flyweight_factory = FlyweightFactory(cars)
    flyweight_factory.display_flyweights()
    flyweight = flyweight_factory.get_flyweight(['Ferrari', 'SF90_Stradale'])
    flyweight.operation(['CL234IR', 'John_Doe'])
    flyweight = flyweight_factory.get_flyweight(['McLaren', 'Senna'])
    flyweight.operation(['CL234IR', 'John_Doe'])
    flyweight_factory.display_flyweights()

if __name__ == '__main__':
    main()  # List of flyweights — ('Ferrari SF90 Stradale', 'McLaren Speedtail', 'SSC Tuatara')
            # Reusing existing flyweight
            # Shared state — ("Ferrari", "SF90 Stradale")
            # Extrinsic state — ("CL234IR", "John Doe")
            # Creating new flyweight
            # Shared state — ("McLaren", "Senna")
            # Extrinsic state — ("CL234IR", "John Doe")
            # List of flyweights — ('Ferrari SF90 Stradale', 'McLaren Speedtail',
                                    'SSC_Tuatara', 'McLaren_Senna')
```

Flyweight Pattern: Pros

- Memory will be saved, assuming your program has a lot of similar objects

Flyweight Pattern: Cons

- Trading memory over CPU cycles when context data needs to be recalculated

Practical

- Series of tasks covering today's lecture
- Worth 1% of your final mark for the Object-Oriented Systems Development course
- Deadline: Tuesday, 7 April at 5pm

Exam 03

- Series of tasks covering lectures 09-12
- Worth 6% of your final mark for the Object-Oriented Systems Development course
- Deadline: Thursday, 2 April at 5pm

Lecture 13: Template Pattern Topics

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