



# Lecture 12: Flyweight Pattern

## IN710: Object-Oriented Systems Development

### Semester One, 2020

Kaiako: Grayson Orr

Te Kura Matatini ki Otago, Ōtepoti, Aotearoa

Thursday, 16 April

# LECTURE 11: STATE PATTERN RECAP

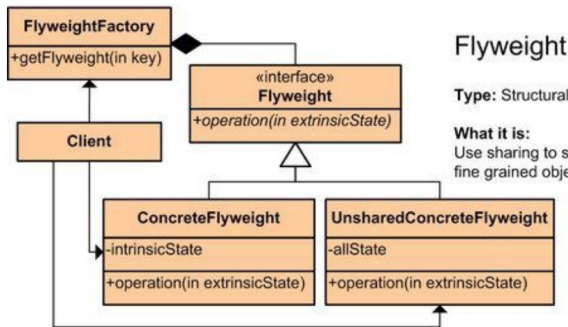
- ▶ Design pattern 07: state pattern
  - ▶ Definition
  - ▶ Problem/solution
  - ▶ Real world analogy
  - ▶ UML & implementation
  - ▶ Pros & cons

# LECTURE 12: FLYWEIGHT PATTERN TOPICS

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

# FLYWEIGHT PATTERN: GoF

## ► GoF definition & UML



## Flyweight

Type: Structural

### What it is:

Use sharing to support large numbers of fine grained objects efficiently.

# 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

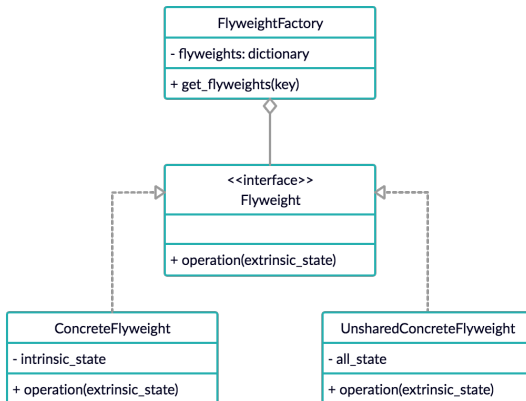
- ▶ Word processor application
- ▶ Create an object for each character typed
- ▶ Contain information such as font face, font size, etc
- ▶ A document might contain tens of thousands of characters

# 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

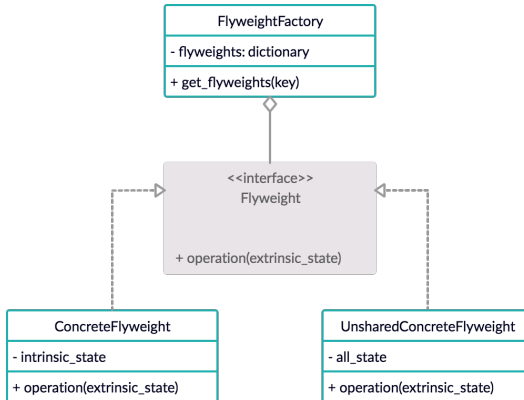
- Consider the following UML diagram:





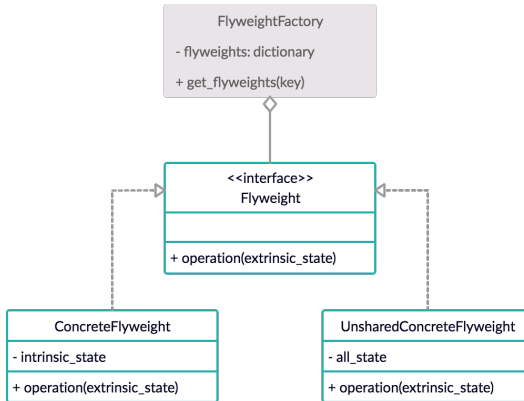
# FLYWEIGHT PATTERN: UML

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



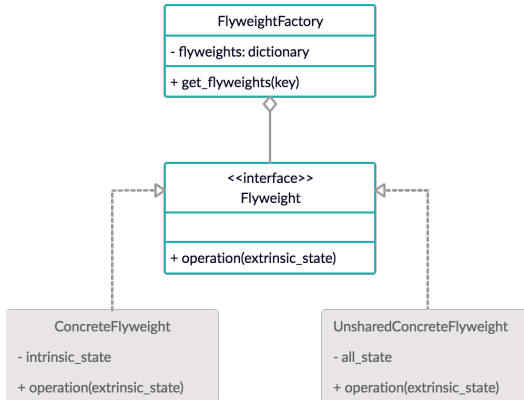
# FLYWEIGHT PATTERN: UML

- ▶ 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



# 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

# 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 13: TEMPLATE PATTERN TOPICS

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