Lecture 09: Adapter Pattern
IN710: Object-Oriented Systems Development
Semester One, 2020

Kaiako: Grayson Orr

Te Kura Matatini ki Otago, Ōtepoti, Aotearoa

Tuesday, 17 March

Lecture 08: Singleton Pattern Recap

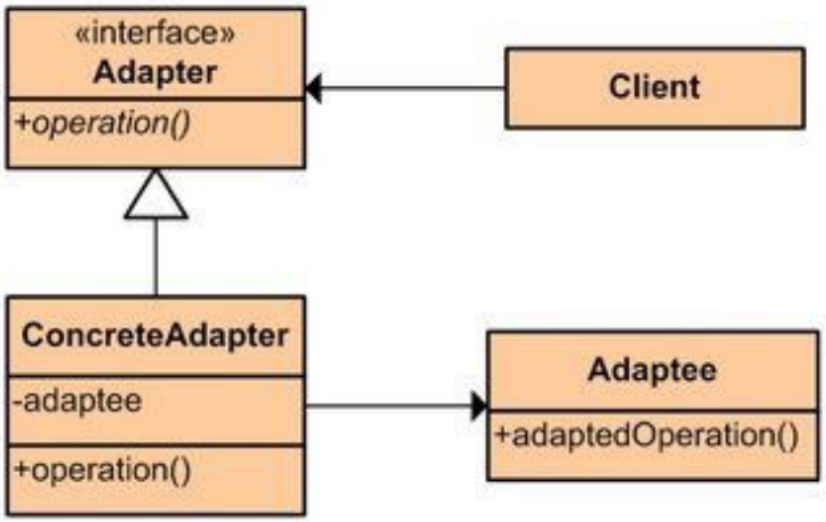
- Design pattern 04: singleton pattern
  - Definition
  - Problem/solution
  - Real world analogy
  - UML & implementation
  - Pros & cons

Lecture 09: Adapter Pattern Topics

- Design pattern 05: adapter pattern
  - Definition
  - Problem/solution
  - Real world analogy
  - UML & implementation
  - Pros & cons

Adapter Pattern: GoF

- GoF definition & UML



Adapter

Type: Structural

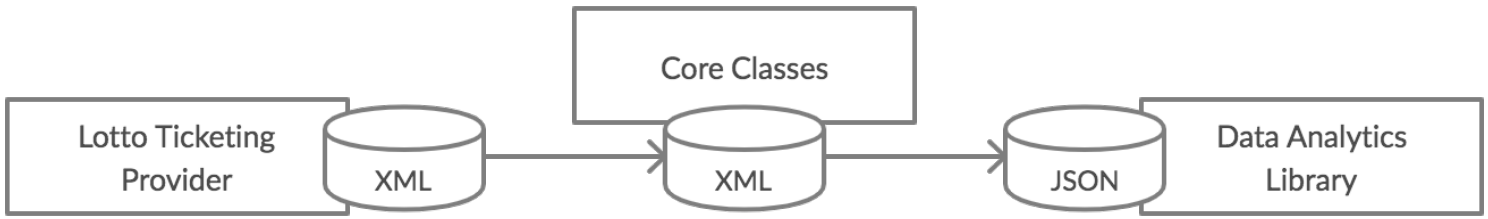
What it is:
Convert the interface of a class into another interface clients expect. Lets classes work together that couldn't otherwise because of incompatible interfaces.

Adapter Pattern: Definition

- Structural pattern
- Also known as wrapper
  - An alternative name shared with the decorator pattern
- Allows the interface of an existing class to be used as another interface
- Often used to make existing classes work with others without modifying their code

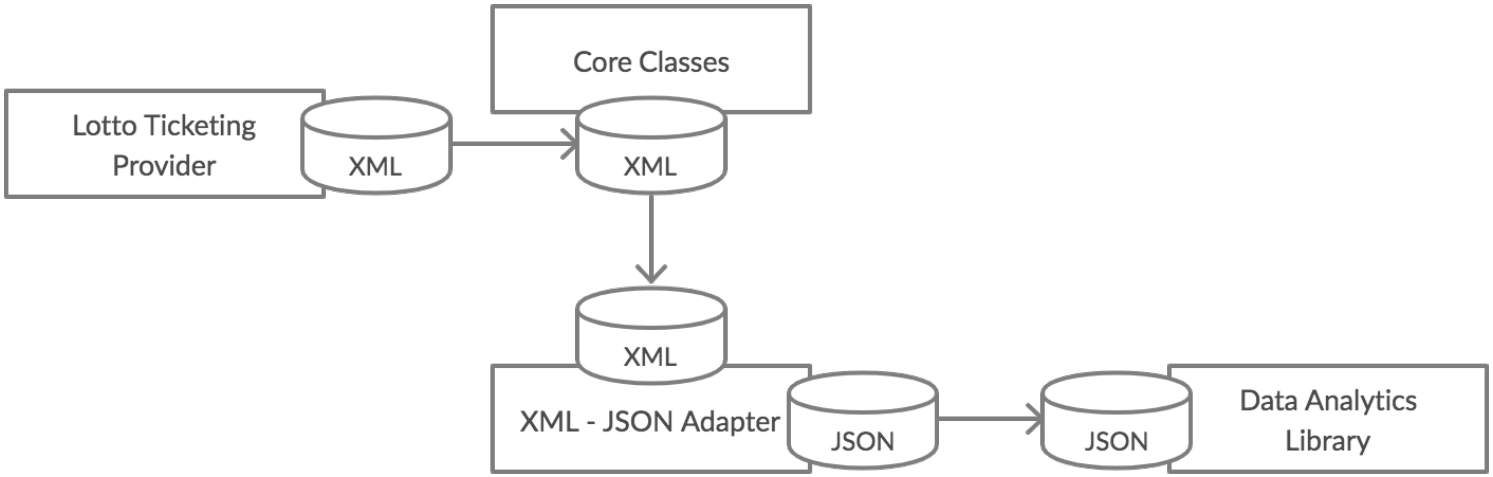
Adapter Pattern: Problem

- Lotto ticketing application



Adapter Pattern: Solution

- Create an XML to JSON adapter

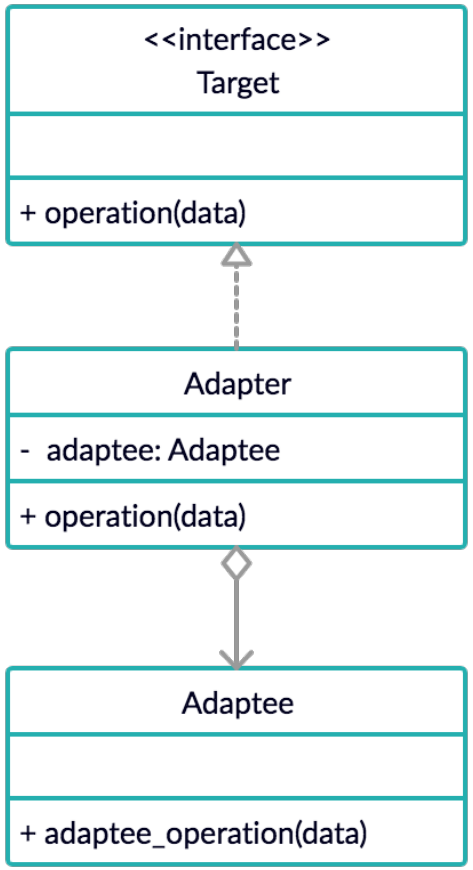


Adapter Pattern: Real World Analogy

- Power plugs & sockets
- Standards are different in different countries
- Example: a US power plug may not fit in a European socket
- The problem can be solved by using a travel adapter

Object Adapter Pattern: UML

- Consider the following UML diagram:

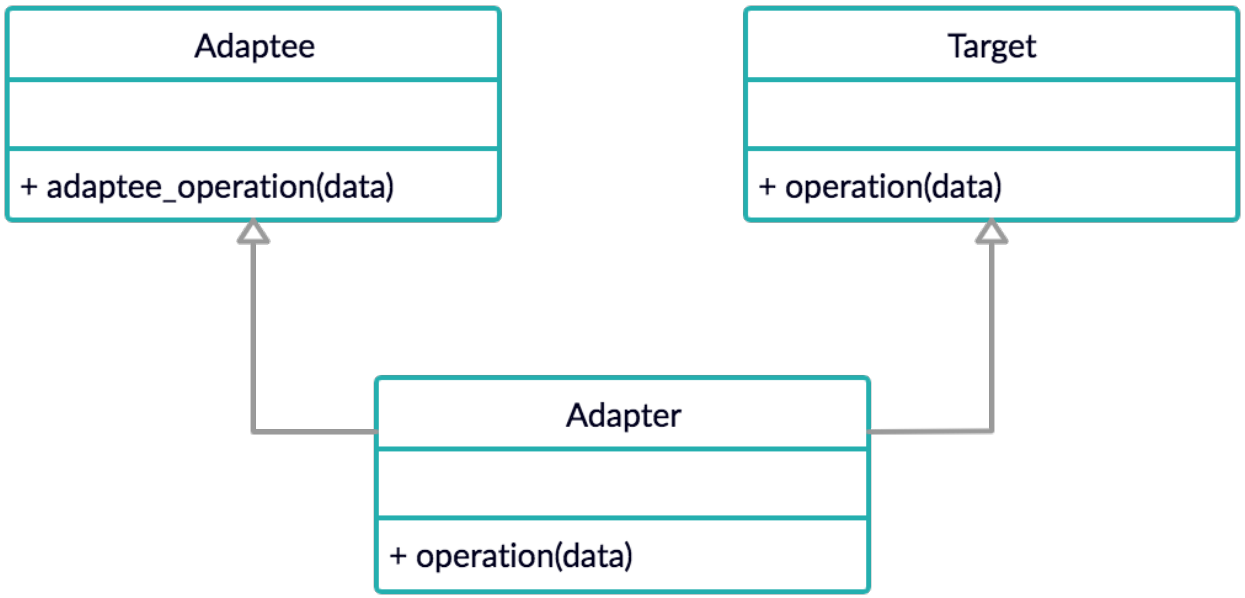


Object Adapter Pattern

- Contains an instance of the class it wraps
- Makes calls to the instance of the wrapped object

Class Adapter Pattern: UML

- Consider the following UML diagram:



Class Adapter Pattern

- Multiple polymorphic interfaces
- Implement interfaces from both objects at the same time
- Can only be implemented in programming languages that support multiple inheritance

Adapter Pattern: Implementation

```
from abc import ABC, abstractmethod

class EuropeanSocket(ABC):
    @abstractmethod
    def voltage(self):
        pass

    @abstractmethod
    def live(self):
        pass

    @abstractmethod
    def neutral(self):
        pass

    @abstractmethod
    def earth(self):
        pass

class USSocket(ABC):
    @abstractmethod
    def voltage(self):
        pass

    @abstractmethod
    def live(self):
        pass

    @abstractmethod
    def neutral(self):
        pass
```

Adapter Pattern: Implementation

```
class AdapteeSocket(EuropeanSocket):
    def voltage(self):
        return 220

    def live(self):
        return 1

    def neutral(self):
        return -1

    def earth(self):
        return 0

class Adapter(USSocket):
    def __init__(self, socket):
        self.socket = socket

    def voltage(self):
        return 110

    def live(self):
        return self.socket.live()

    def neutral(self):
        return self.socket.neutral()
```

Adapter Pattern: Implementation

```
class FlatIron:
    def __init__(self, adapter):
        self.adapter = adapter

    def heating(self):
        if self.adapter.voltage() > 110:
            print('Your flat iron is overheating')
        elif self.adapter.live() == 1 and self.adapter.neutral() == -1:
            print('Your hair is ready to be straighten')
        else:
            print('The power is off')

def main():
    adaptee_socket = AdapteeSocket()
    adapter = Adapter(adaptee_socket)
    flat_iron = FlatIron(adapter)
    flat_iron.heating()

if __name__ == '__main__':
    main()  # Your hair is ready to be straighten
```

Adapter Pattern: Pros

- Separate the interface from the primary business logic of the program
- New types of adapters can be introduced without having to change the client's code

Adapter Pattern: Cons

- Overall, the complexity of the code increases - new interfaces & classes need to be introduced

Practical

- Series of tasks covering today's lecture
- Worth 1% of your final mark for the Object-Oriented Systems Development course
- Deadline: Tuesday, 31 March at 5pm

Reminder: Exam 02

- Series of tasks covering lectures 05-08
- Worth 6% of your final mark for the Object-Oriented Systems Development course
- Deadline: Thursday, 19 March at 5pm

Lecture 10: Builder Pattern Topics

- Design pattern 06: builder pattern
  - Definition
  - Problem/solution
  - UML & implementation
  - Applicability
  - Pros & cons