COMP2511

Tutorial 5

Last Week's Tutorial

- Code Smells
- SOLID Principles
- Streams
- Design by Contract

This Week's Tutorial

- Design Patterns
  - Introduction
- Behavioural Design Patterns
  - Strategy Pattern
  - Observer Pattern
  - State Pattern

Notices

- Assignment 1 is due this Friday, 3pm.
  - Make sure that your code compiles on CSE (!!!)
- Assignment 2 will be released this week.
  - Groups and channels on Teams have been created, please inform of any issues!
  - You are free to communicate on any platform that you prefer, but if any issues arise in terms of participation, cooperation etc. I can only infer off the information on Teams.
- Flex Week next week!
  - Try to find time to unwind so that you're ready for the next half of the term!
  - Lab 5 will be due on **Week 7 Monday**.
  - Flex Week will be a good time to start communicating with your partner to figure out a plan of attack for Assignment 2 according to each of your availabilities.
- I will be putting out an anonymous feedback form where you can share any thoughts on how my tutorials have gone so far. Any responses will be appreciated! This feedback will help me curate my tutorials to your needs and preferences.

Design Patterns

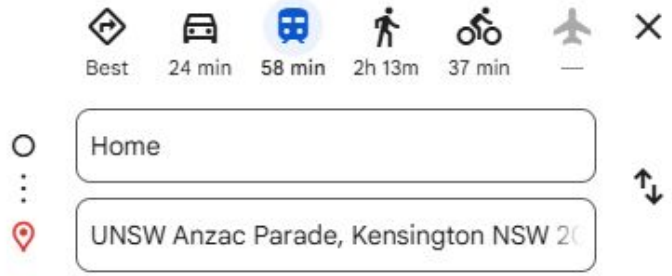
- Design patterns are typical **repeatable** solutions to common problems in software design.
  - They are basically the ‘tried and true’ solutions to specific problems, and are widely recognisable to other programmers.
- They are not immediate solutions themselves - they represent **templates** that you need to know how to *apply* correctly to specific scenarios.
- **Refactoring Guru**: a great resource that has explanations and examples for each of the design patterns covered in this course.
- Don’t use patterns for the sake of using patterns - always consider if using a design pattern makes sense and is a good solution to your specific problem.

Types of Design Patterns

- **Behavioural** patterns (this week's tutorial)
  - Concerned with how classes actually do things and the assignment of responsibilities between objects.
- **Structural** patterns
  - Concerned with how to assemble objects and classes into larger structures while keeping these structures flexible and efficient.
- **Creational** patterns
  - Concerned with introducing different ways to create more complicated objects.

Strategy Pattern

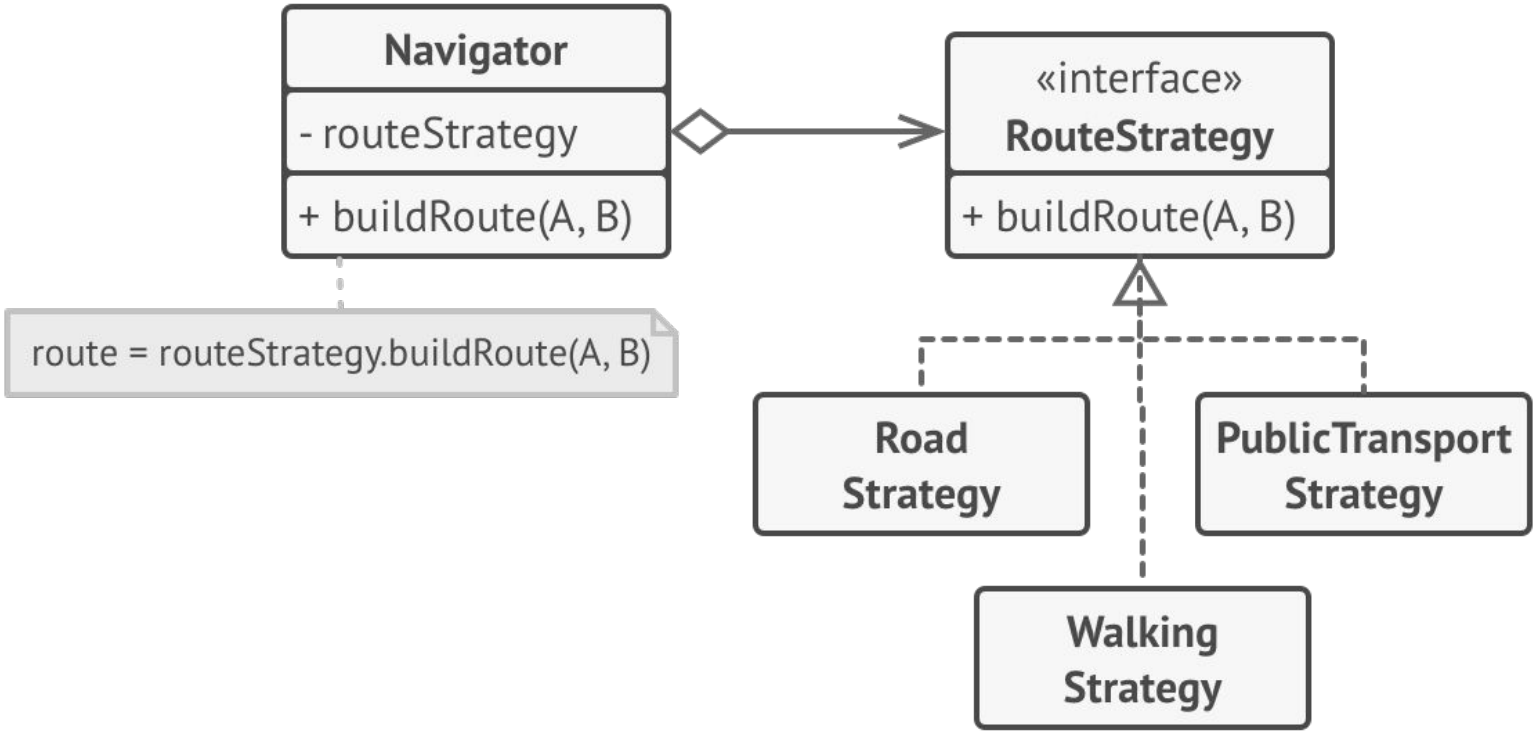
- Consider how you have multiple options on Google Maps for your mode of transport to get to UNSW - for example, you could walk, drive or take public transport.
    - Regardless of which mode of transport you use, your overall goal is to arrive at UNSW. The specific details of the route you have to take, the amount of distance you have to cover, the costs incurred etc. differ depending on which mode of transport you use.
    - Also, you can compare between the different modes of transport at will.



Strategy Pattern

- The strategy pattern is a behavioural design pattern that allows classes to switch between specific implementations at **run-time**.
- Rather than putting all of the logic of the different implementations in a single class, separate this logic into other classes called *strategies*.
  - Each strategy should implement a specific **interface/abstract class** that defines the methods that every strategy should be able to perform.
  - The original class then stores **instances** of these strategies, and executes instructions according to the interface methods of those strategies.
  - Switching between strategies can be done through either a setter or switch statements (if input is required from sort of user interface), which is ok since this would be comparably much less switching than before.

Strategy Pattern Implementation



Live Example

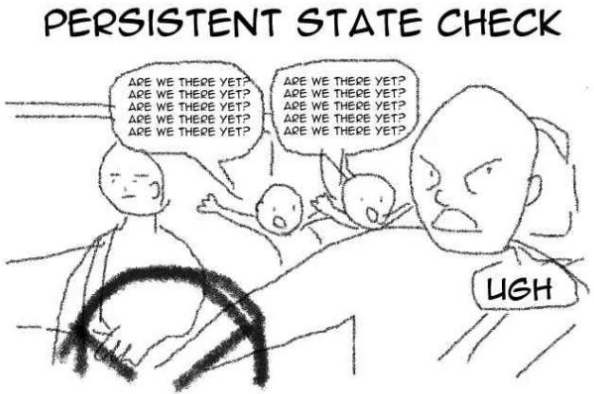
src/restaurant (details in README.md)

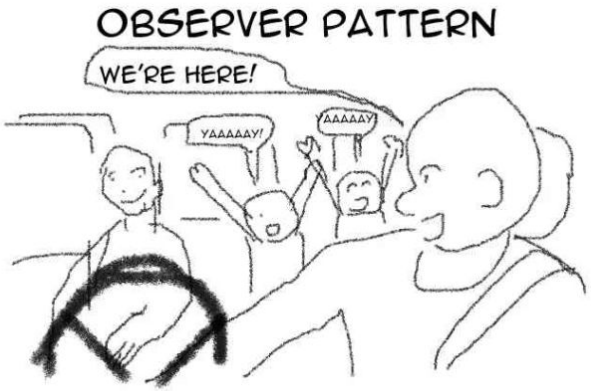
Observer Pattern

- If you were subscribed to some YouTube channel and really wanted to watch videos as soon as they came out, would you rather:
  - Keep refreshing the YouTube channel every 5 seconds to see if they have uploaded a new video, or
  - Receive a notification whenever a new video has been uploaded?
- The more sensible option (and how it works in reality) is the second one! When a YouTuber publishes a new video, their subscribers get notified.
- We can frame this in a specific way: The subscribers wait on some important **event** (in this instance, a new video being uploaded), and when that event happens, some actions are performed to **respond** to it (in this instance, a notification is sent).

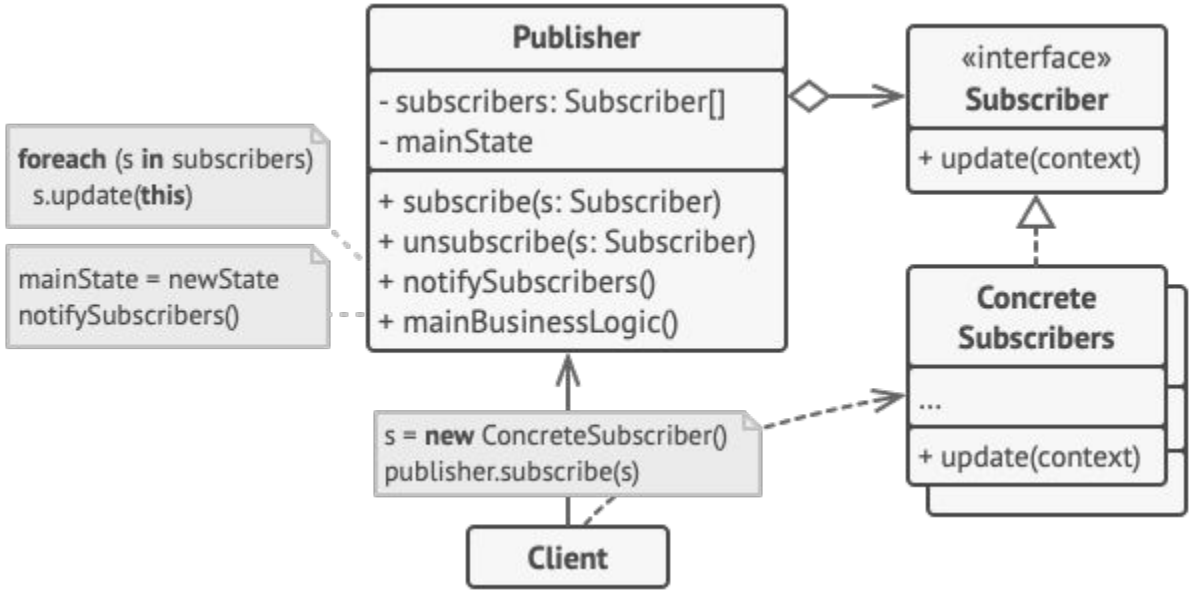
Observer Pattern

- The observer pattern is a behavioural design pattern where a **subject** maintains a list of **observers**, who are **notified** when some event occurs.
  - Upon being notified, the observers will perform some action which is usually related to some sort of data that the subject sends out.





Observer Pattern Implementation



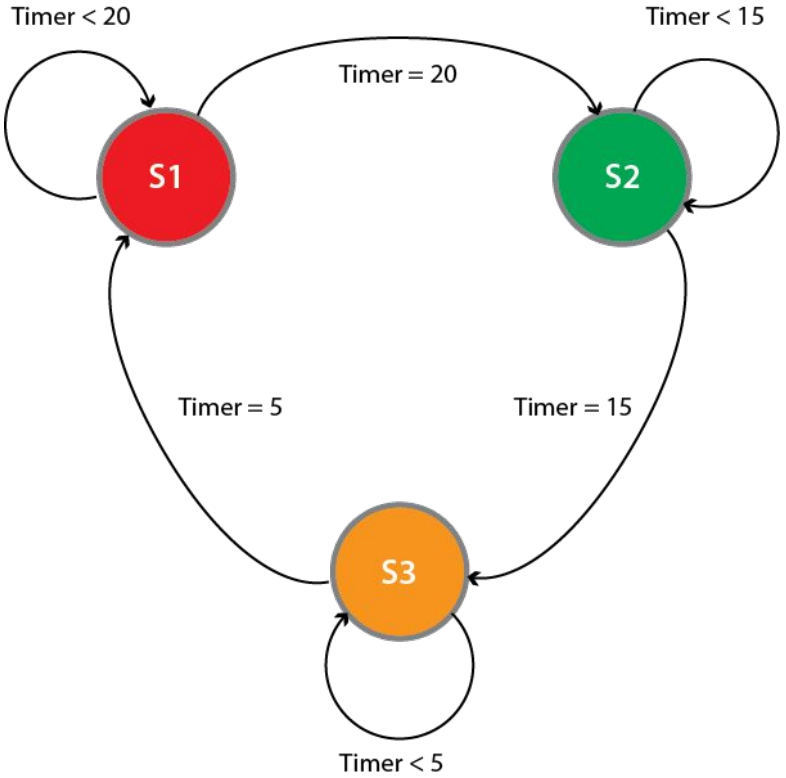
Live Example

src/youtube (details in README.md)

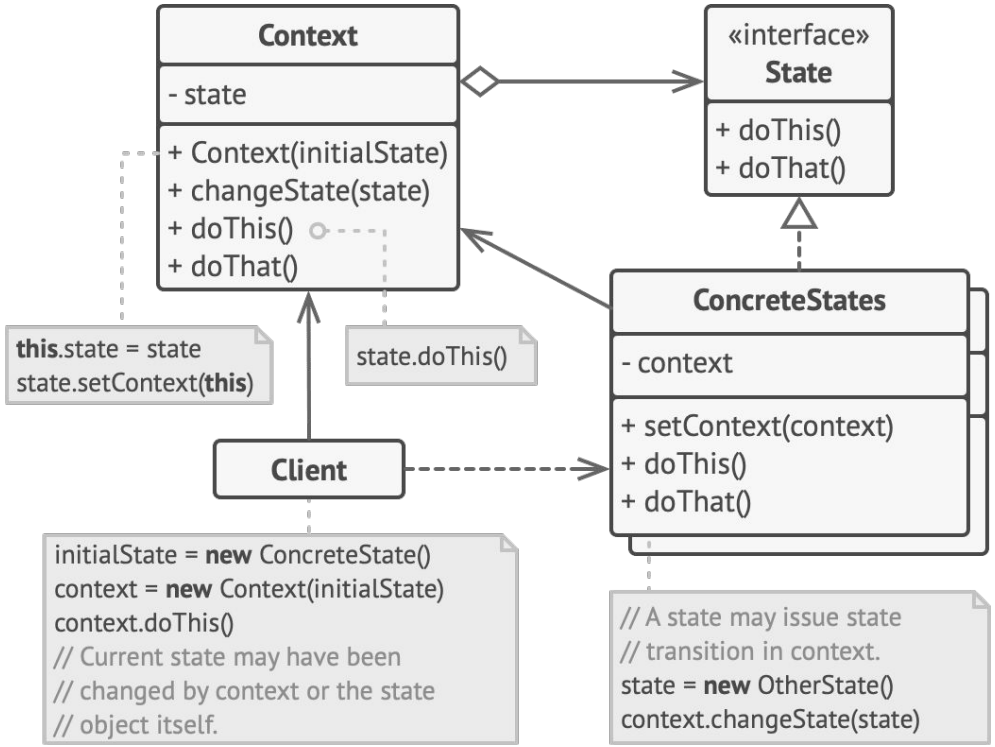
State Pattern

- The state pattern is a behavioural design pattern that is applicable when an object has clearly defined **states** that can be moved between, and the behaviour of the object is mostly or wholly dependent on whichever state it is in.
  - Essentially, we want the object's behaviour to change whenever its internal state changes, and we would like the object to manage its transitions between states by itself.
  - If an object has such states and transitions between them, a **state diagram** that represents the different states and how to switch between them can be made.
- For example, a traffic light can be either red, yellow or green. It switches between these states every minute or so, and the meaning of the light is entirely dependent on whichever colour is currently showing.

State Diagram



State Pattern Implementation



Code Example

src/video (details in README.md)