COMP2511

Tutorial 5

Week 5 Notices

- Assignment 1 is due today!
  - You can run **2511 dryrun ass1** on CSE to check that your code compiles.
- Assignment 2 has been released!
  - Groups will be shown on Teams and channels for each group will be created.
  - You are free to communicate on any platform that you prefer, but if any issues arise in terms of participation, cooperation etc. I will have to go off of what is on Teams.
- Flex Week!
  - Try to find time to unwind, and make sure you're caught up for the second half of the term.
  - Lab 5 will be due on **Week 7 Monday** - there's quite a bit to do, but please don't skip it!
  - Start communicating with your partner to figure out your plan for Assignment 2.

This Week's Tutorial

- Design Patterns
- Strategy Pattern
- Observer Pattern
- State Pattern

Design Patterns

- Design patterns are typical **repeatable** solutions to common problems in software design.
  - They are basically the ‘tried and true’ solutions, 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.
- 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.
- Shoutout: **Refactoring Guru**, a great resource for this course that explains and has examples for each of the design patterns covered.

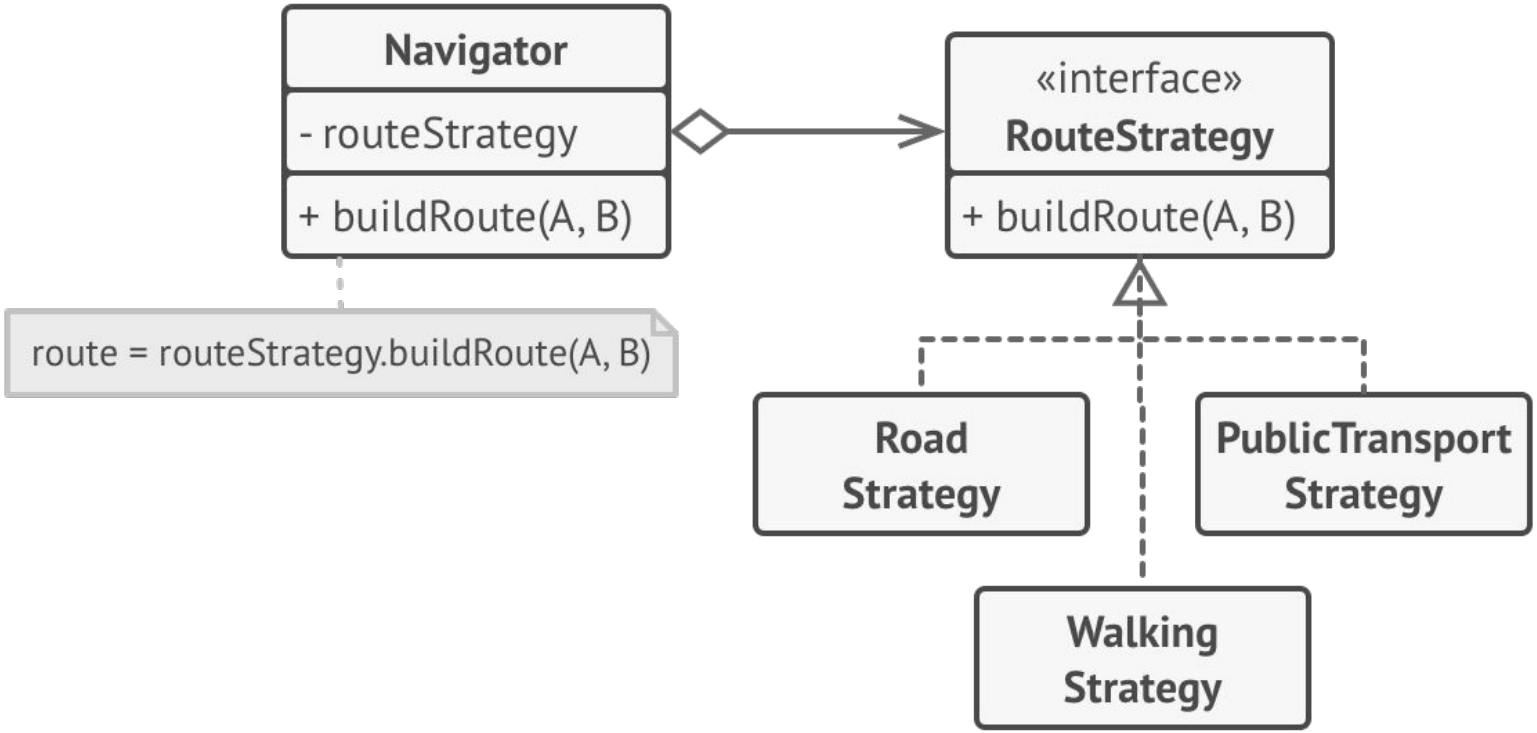
Types of Design Patterns

- **Behavioural** patterns (the focus of 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 the construction of specific objects.

Strategy Pattern

- The strategy pattern is a **behavioural** design pattern that allows classes to switch specific implementations at run-time.
- An example of something that could use many different implementations is a sorting algorithm - for example, you could use merge sort, quick sort, heap sort, etc.
- Rather than putting all of the logic of the different implementations in a single class, separate them into other classes and execute your instructions according to a specific interface that each strategy will **implement**.

Strategy Pattern Implementation



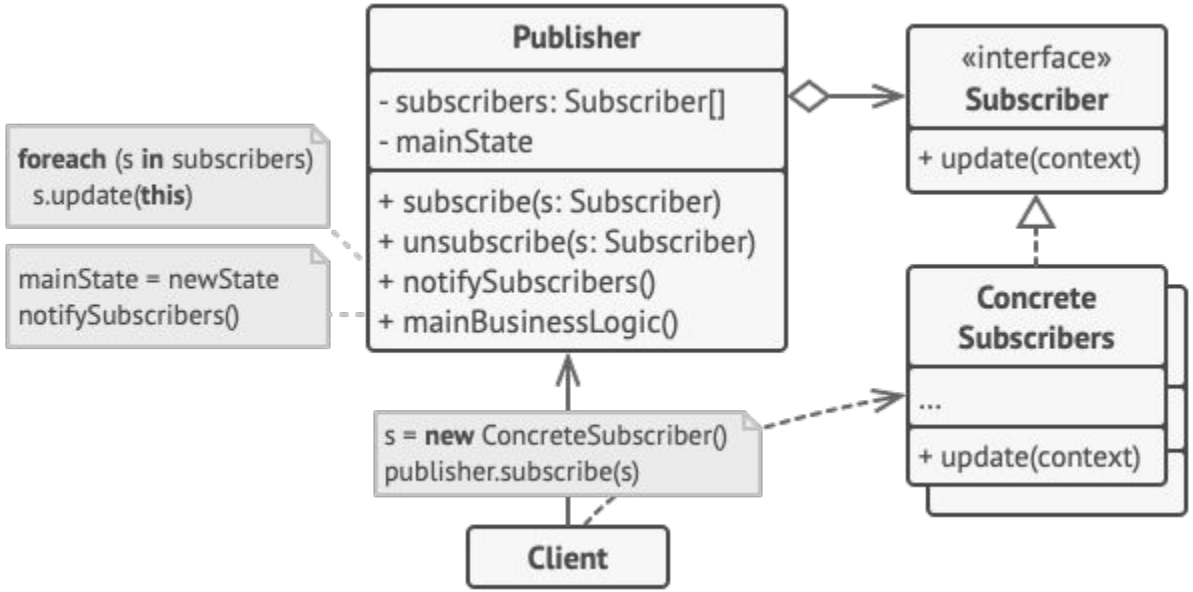
Code Example

src/restaurant

Observer Pattern

- The observer pattern is a **behavioural** design pattern that allows a **subject** to automatically **notify** its **observer(s)** of any important events.
- You can think of this like a YouTuber and their subscribers - the YouTuber is the subject, and their subscribers are the observers that are immediately notified when a new video or post is published (if they have notifications on, that is!)

Observer Pattern Implementation



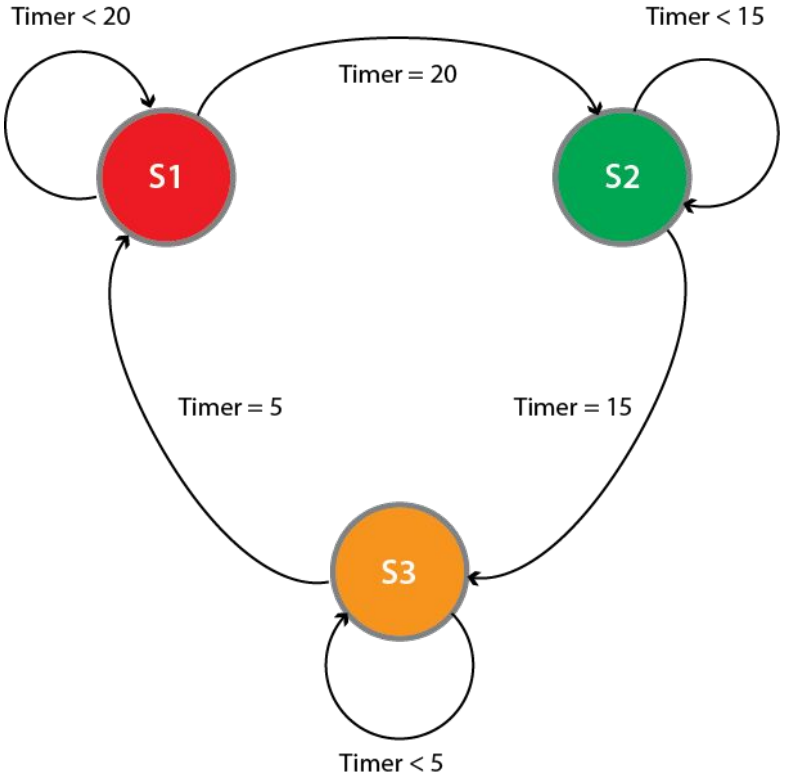
Code Example

src/youtube

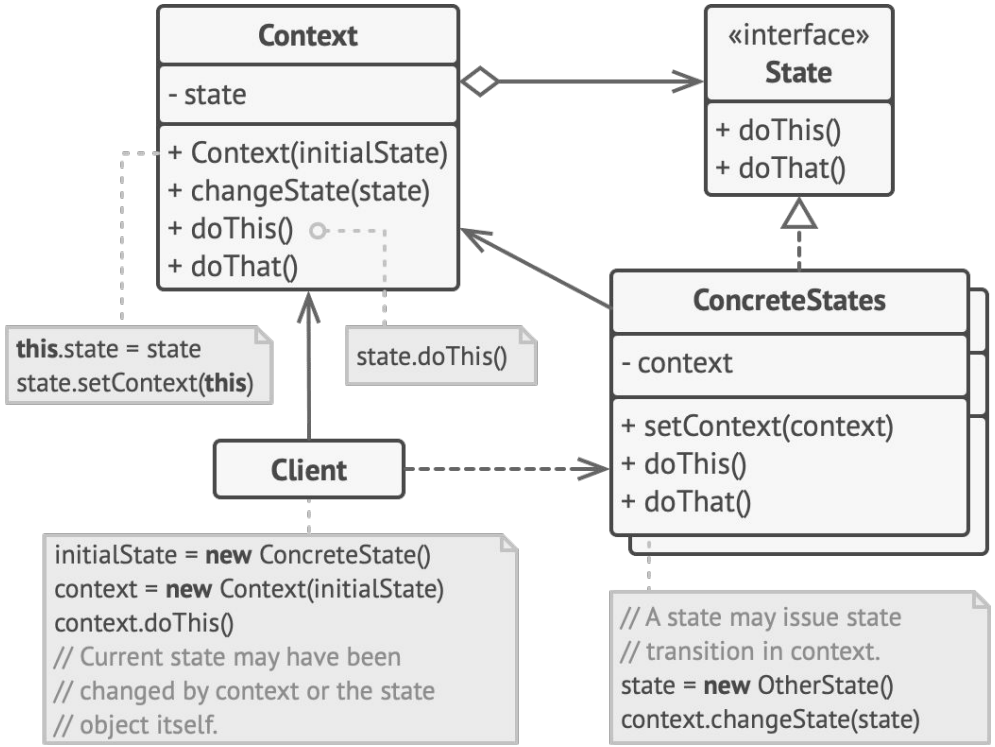
State Pattern

- The state pattern is a **behavioural** design pattern that allows an object to modify its behaviour whenever its internal **state** changes, and functionality is specific to whichever state an object is currently in.
- This makes sense for objects that have clearly defined states that can be moved between, like a traffic light switching between a red, yellow and green light.

State Diagram



State Pattern Implementation



Code Example

src/youtube2