COMP2511

Tute04

Agenda

- Lambdas
- Streams
- Strategy Pattern
- Observer Pattern

```
MyFunctionInterfaceA f1 = (x, y) -> x + y

MyFunctionInterfaceA f2 = (x, y) -> x - y

MyFunctionInterfaceB f3 = (x, y) ->  x > y

MyFunctionInterfaceC f4 = x -> {
                          double y = 1.5*x;
                          return y + 8.0;
                      };

System.out.println( f1.myCompute(10, 20) );
System.out.println( f2.myCompute(10, 20) );
System.out.println( f3.myCmp(10, 20) );
System.out.println( f4.doSomething(10) );
```

Lambdas

Lambdas are “mini functions”, containing:

- parameter list: names only, no types
- arrow token: →
- body: either single expression or statement block

```
ist.stream()
 .filter( e -> e.length() > 0 )
 .mapToInt(String::length)
 .average()
 .getAsDouble();
```

Streams

A sequence of elements, pipelined through some intermediate operations.

- Convert to Stream using nums.stream()
- Pipe operations with “.”, nums.stream().op1().op2()
- Convert back to List using .toList()

The :: Operator

STATIC METHOD

Class::method

- method is static in Class

INSTANCE METHOD

instance::method

- instance of a Class

CLASS CONSTRUCTOR

Class::new

- Calls the Constructor of the Class

App

Strategy Pattern

Strategy Pattern

The common problem?

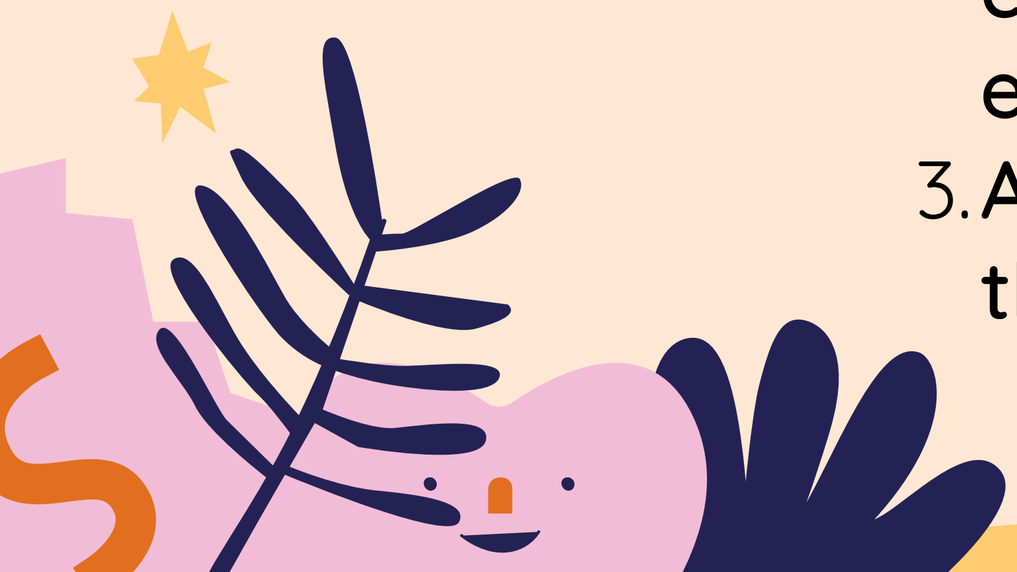
Our object needs to use different variations of an algorithm or display different behaviour, which the object can switch between during runtime:

- Changing modes of transportation on Google Maps (walk vs bike vs car)
- Recieving different payment methods (debit card vs PayPal vs cash)

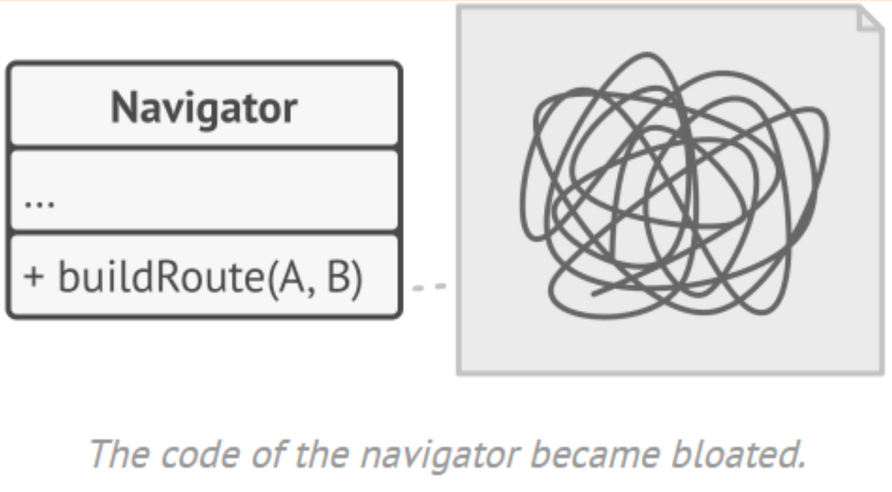
Strategy Pattern

The common solution?

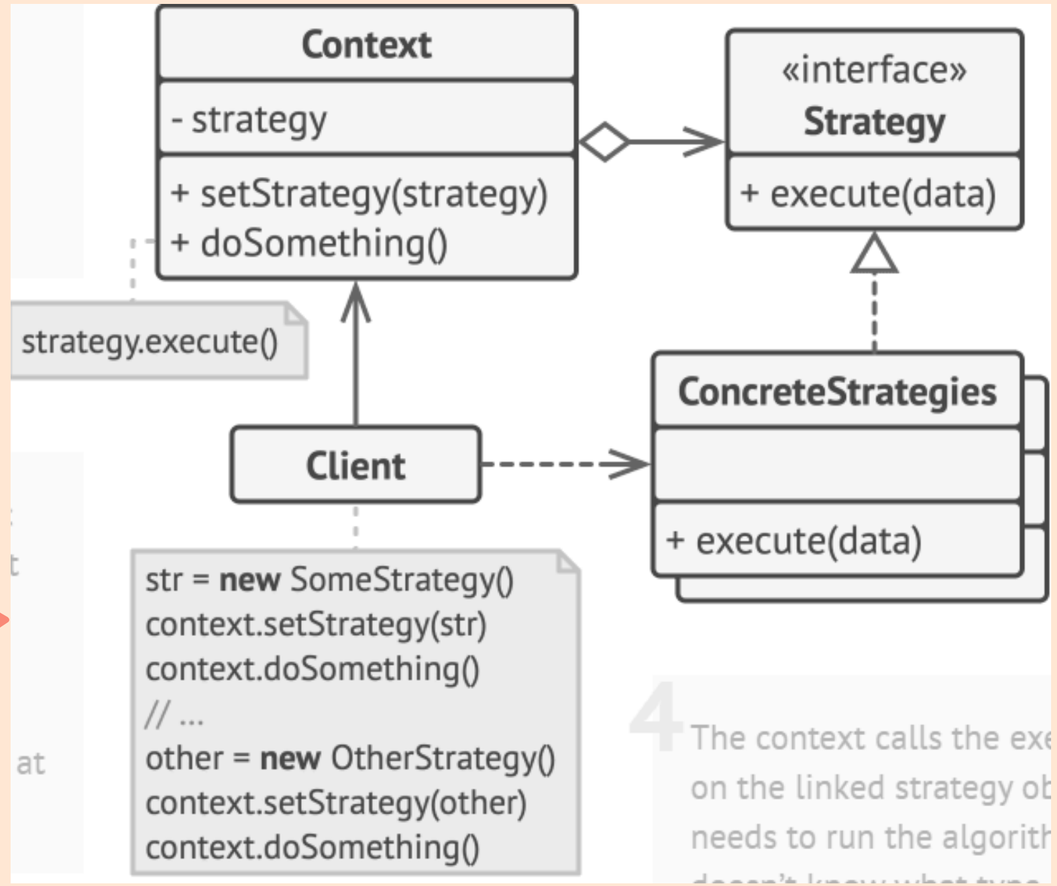
1. Identify the STRATEGIES: different behaviours and algorithms the object can switch between.
2. Create Strategy interface with the common method, implemented by each strategy.
3. Add the interface as an attribute to the object using these strategies.



Strategy Pattern



The code of the navigator became bloated.



Resturant

Design Principles

How is open/closed principle violated?
How does this make the code brittle?

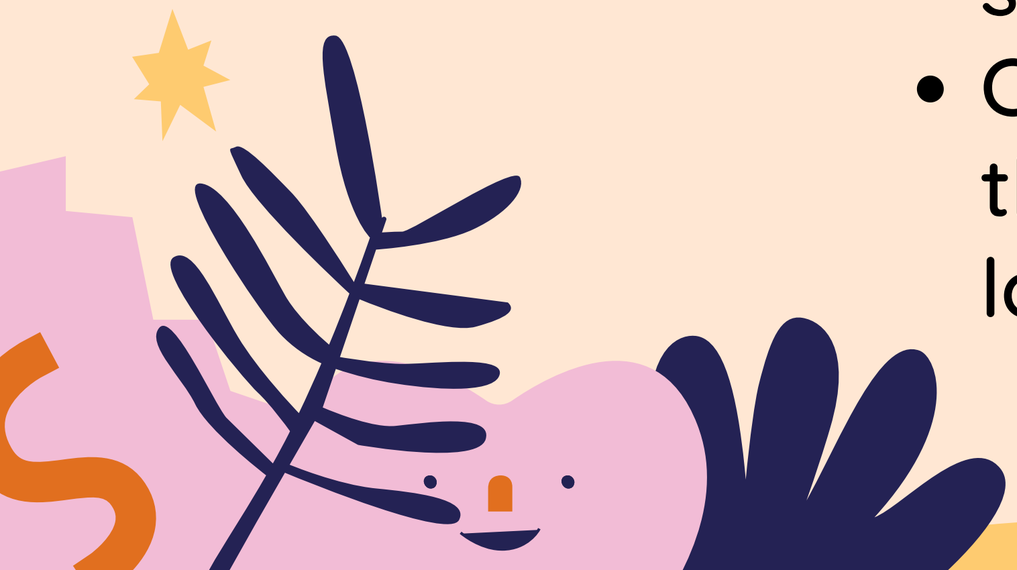
Observer Pattern

Observer Pattern

The common problem?

In our code, some observing objects in our code are waiting for some subject objects to update:

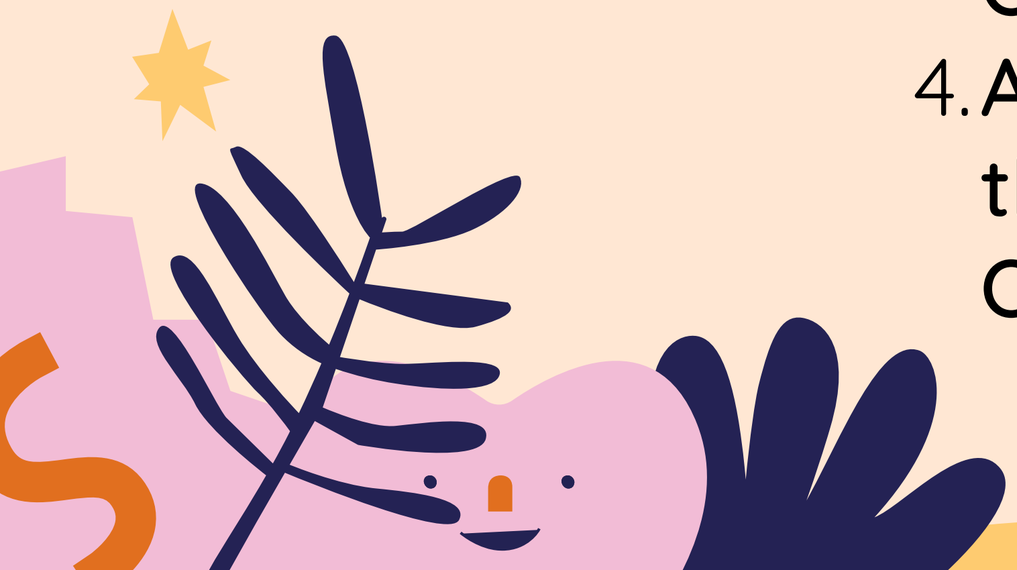
- Customers waiting for their local store to have PlayStation 5s in stock
- Children asking “are we there yet” to their parents every minute instead of looking out of the window to check



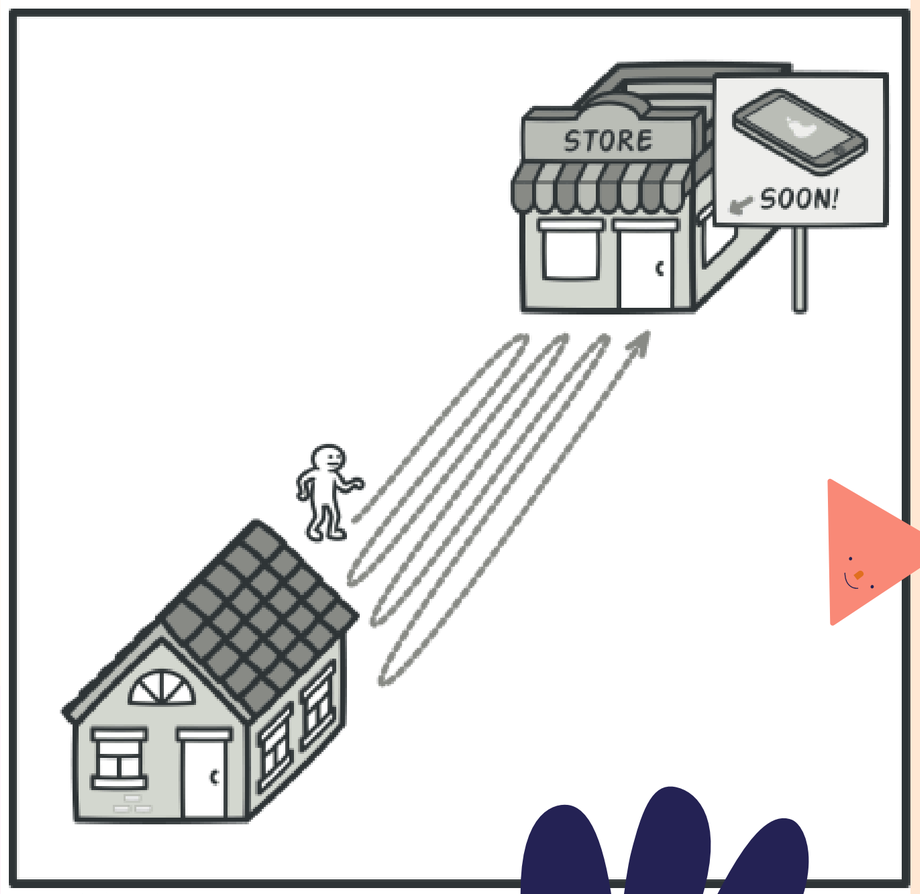
Observer Pattern

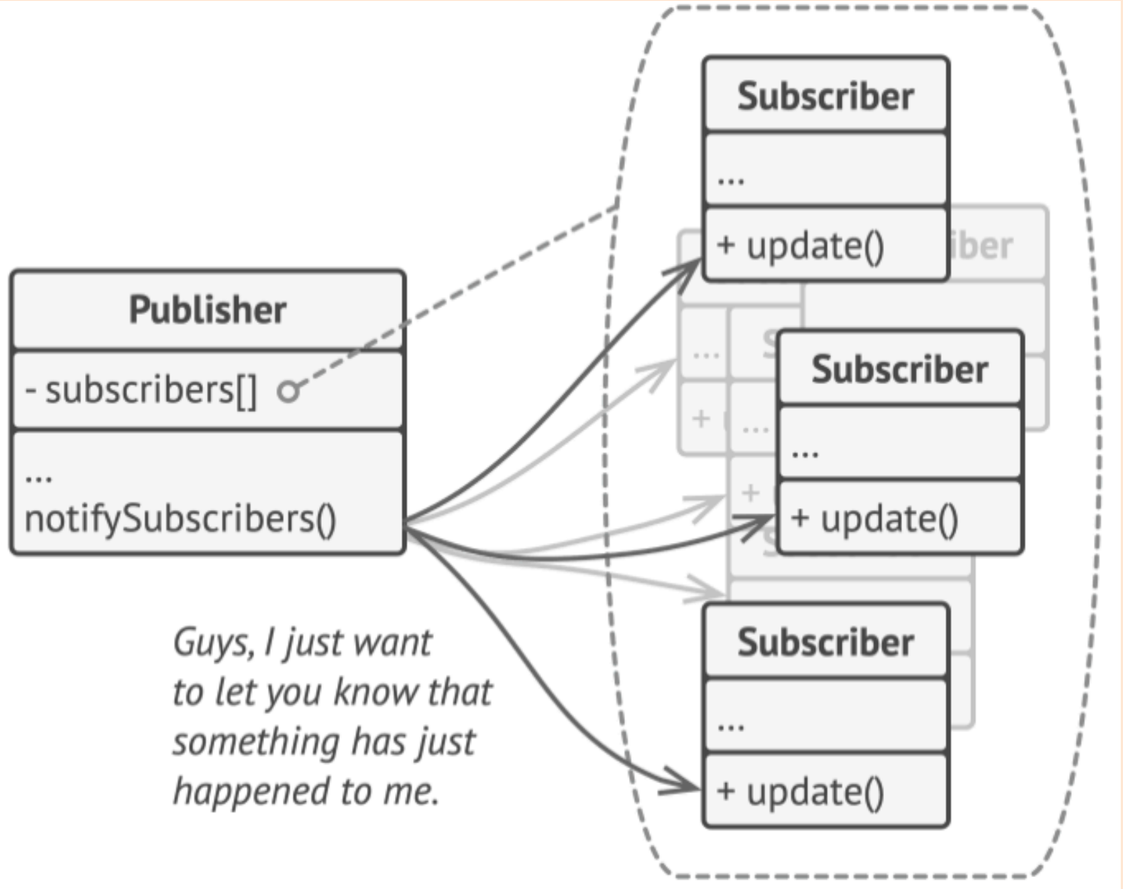
The common solution?

1. Identify the Subjects and Observers.
2. Keep a list of Observers subscribed to the Subject as an attribute.
3. Add an update() method to Observers.
4. Add a notify() method to Subject, that calls all of its subscribed Observer's update() methods.



Observer Pattern





Youtube