Design Patterns

Observations

- There are repeated design structures and roles of classes used in most software
- Not possible to capture these structure and roles into specific classes, i.e., they are a part of a class, but do not make up the entire class
- Language features may support these if organized correctly
- Need a way to communicate about them

Issues

- Must be widely applicable
- Solution must be safe
- Solution should be efficient

Software Design Patterns

- *Design Patterns: Elements of Reusable Object-Oriented Software*
  - Gamma, Helm, Johnson, Vlissides

- - AKA, "Gang of 4 Book" (GOF'95)

Basic Elements of Design Patterns

- Name – how we refer to patterns so others know what we mean
- Problem – what issue are they meant to help with
- Solution – how do we implement the pattern
- Consequences – caveats to consider when applying a pattern

Categories of Design Patterns

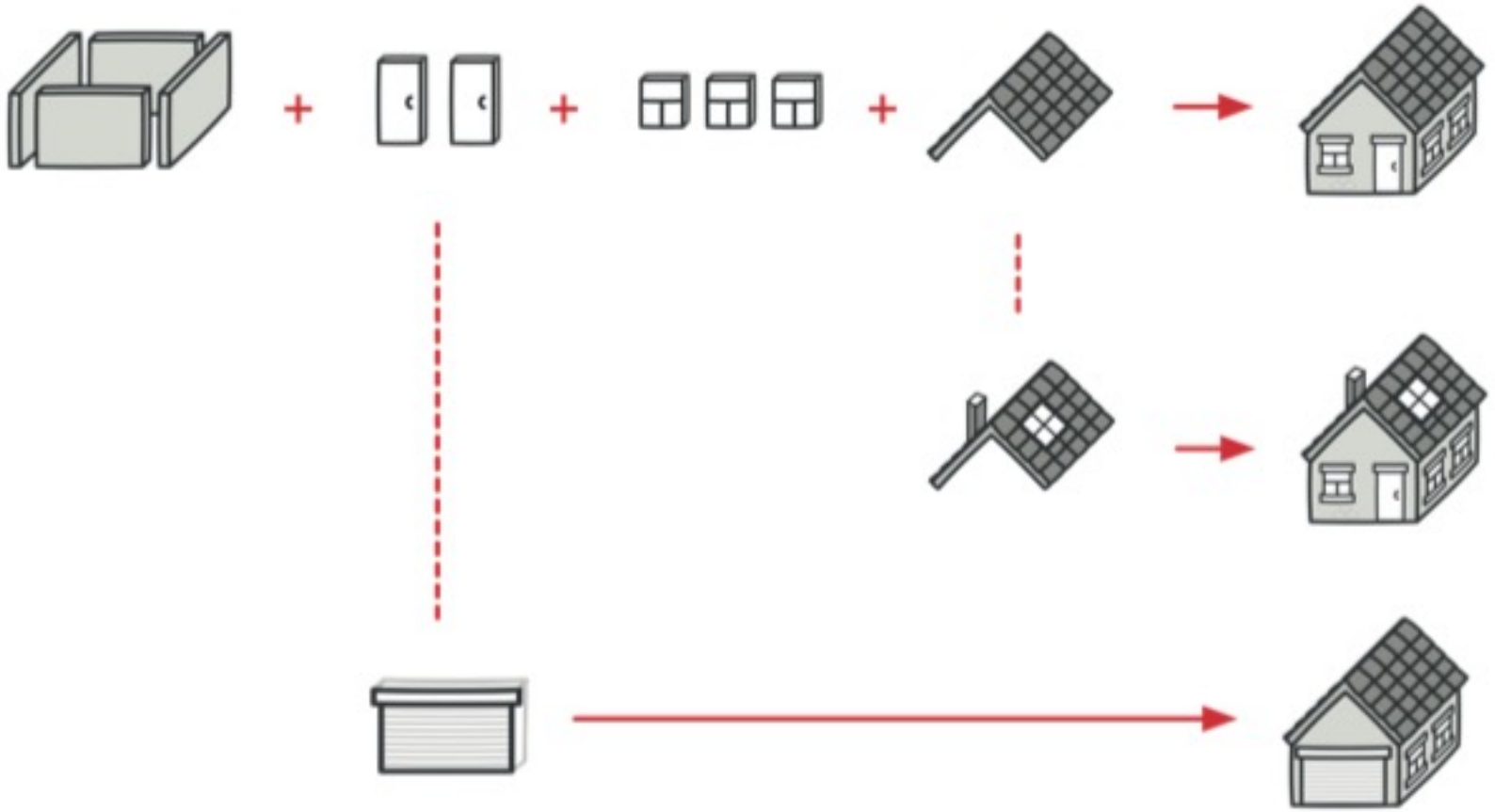
- Creational Patterns
  - provide various object creation mechanisms, which increase flexibility and reuse of existing code
- Structural Patterns
  - explain how to assemble objects and class- es into larger structures, while keeping this structures flexible and efficient.
- Behavioral Patterns
  - concerned with algorithms and the assignment of responsibilities between objects

Template Method

Template Method Pattern

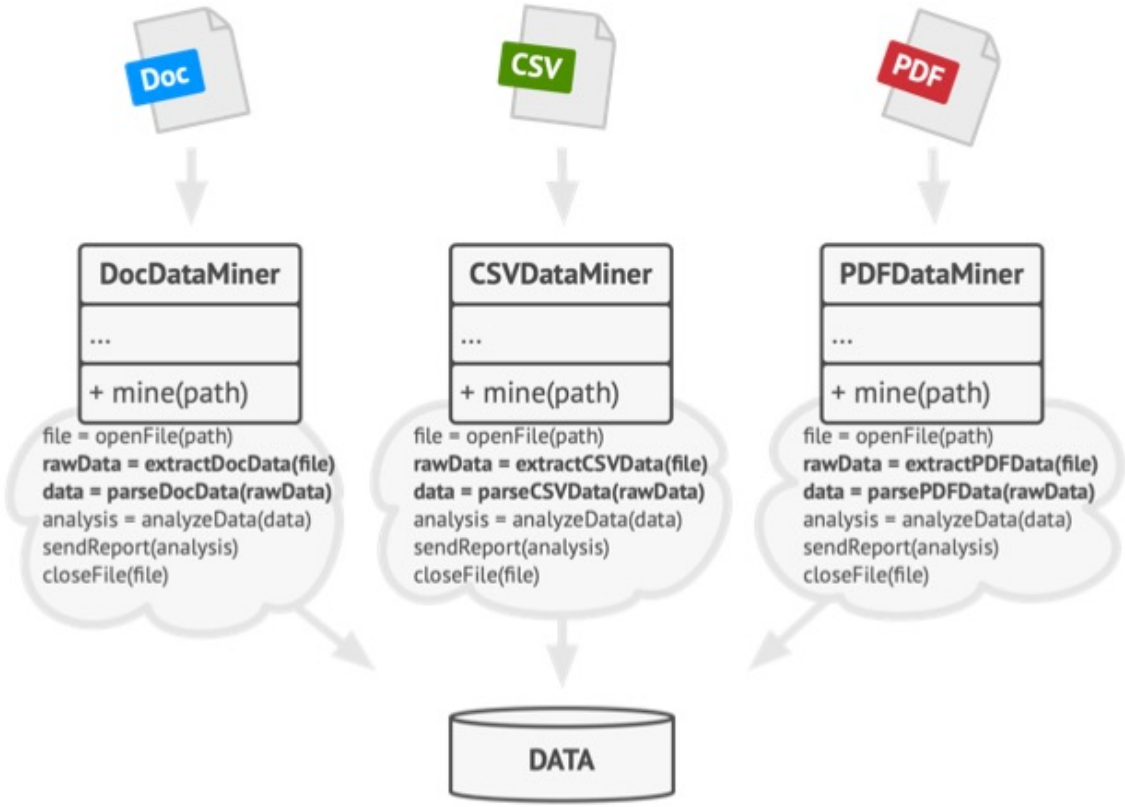
- behavioral design pattern
- defines the skeleton of an algorithm in the superclass
- subclasses override specific steps of the algorithm without changing its structure
- **NOT** related to C++ templates

Analogy



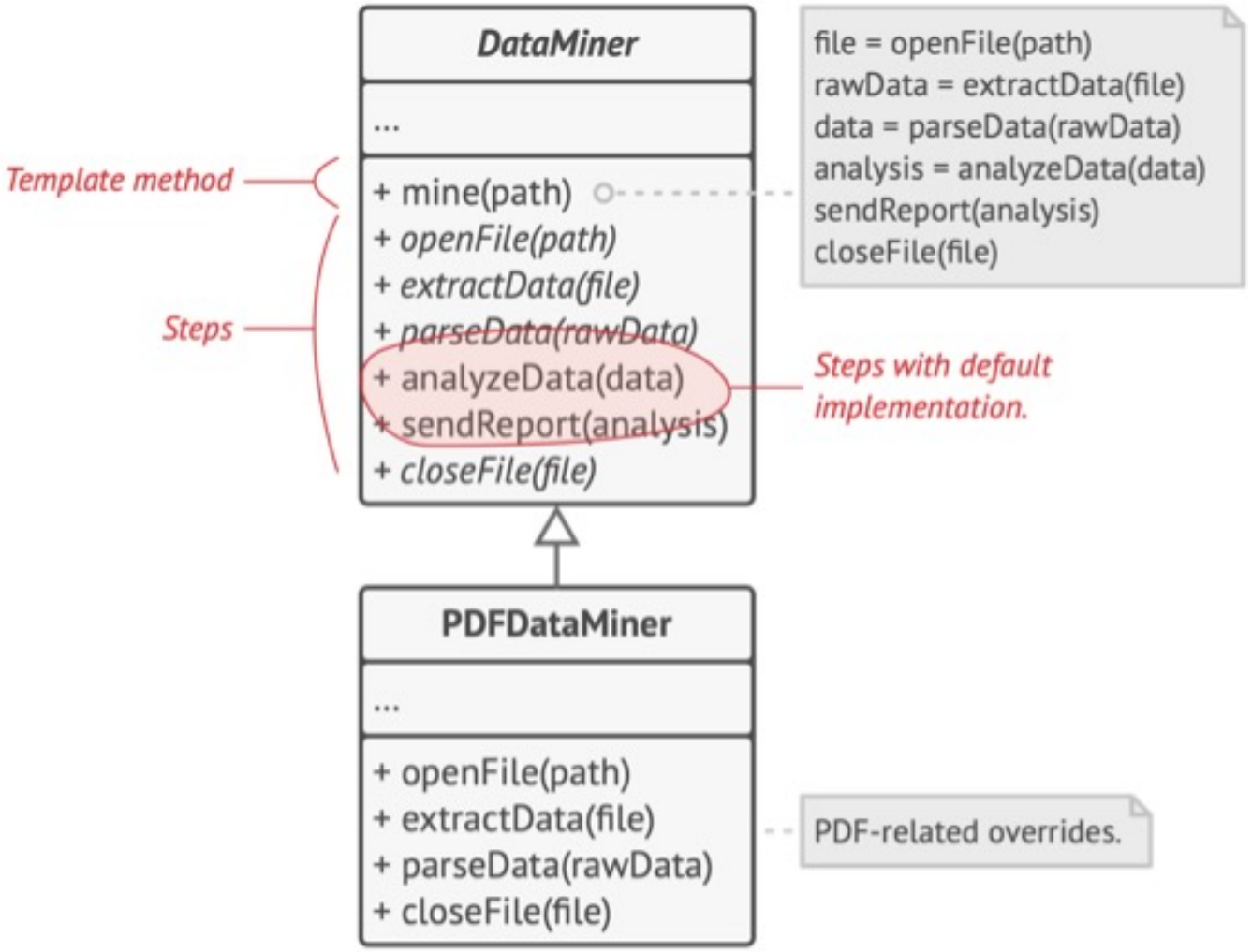
Why?

- Fundamental technique for code reuse
- Particularly important in class libraries to factor out common behavior
- Allows for easy extension to a predefined interface

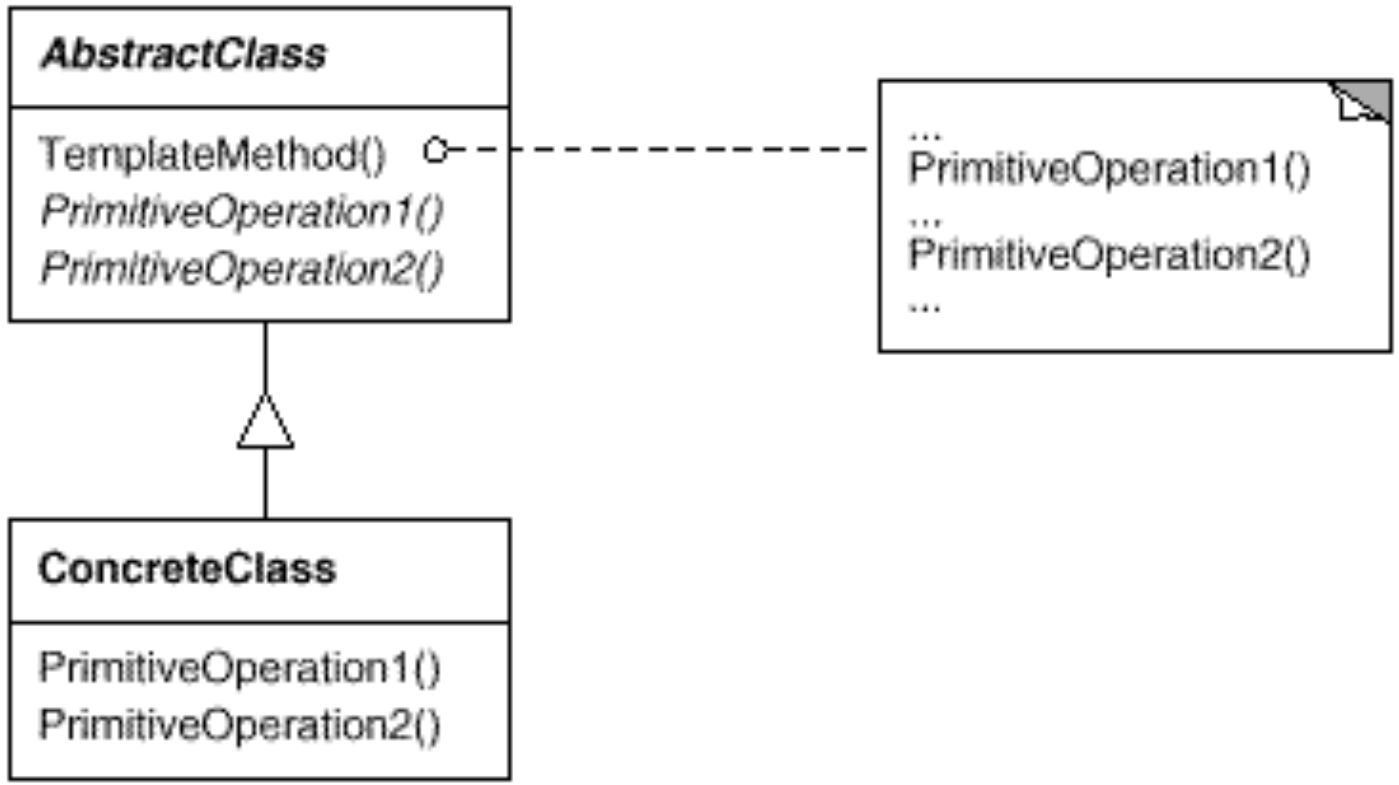


How?

- Implement the *invariant* parts of an algorithm
- Convert the unique steps to be abstract functions
- Some steps can have a default implementation, but can still be overridden
- Add *hooks* before and after crucial parts of the algorithms for extension buy subclasses [optional]



Template Method UML Diagram



Abstract functions are shown in italics in UML

Pros and Cons

+ Client code can override only certain parts of the algorithm, making them less affected by changes to other parts of the algorithm
+ Duplicated code is consolidated to super class

- Might violate Liskov Substitution Principle
- Client code might be limited by the provided steps for an algorithm
- Template methods get more complicated to maintain as the number of steps in the algorithm increase

Observer Pattern

Observer Pattern

- behavioral design pattern

- defines a subscription mechanism to notify multiple objects about any events that happen to the object they’re observing

Real World Analogies

- You are waiting for a graphics card
- Everyday you go to the store and check if there is a graphics card
- Each day you go home sad….
  - until you get one!
- Inefficient!
  - In computing terms, we are “polling” a resource

Real World Analogies

- You want to notify people that you have graphics cards in stock
- You pay an advertising company for the email addresses of everyone in your zip code and send out some ~~spam~~ email
- Some people want a graphics card and are happy…others wonder why you are bothering them
- Inefficient!
  - Closest computing analogy might be a broadcast message to any accessible object/device

Real-World Analogy Solution

- A subscription service
- Indicates your interest in the information
- Sends you messages about the thing you want to know about
- Unsubscribe when you don’t care anymore

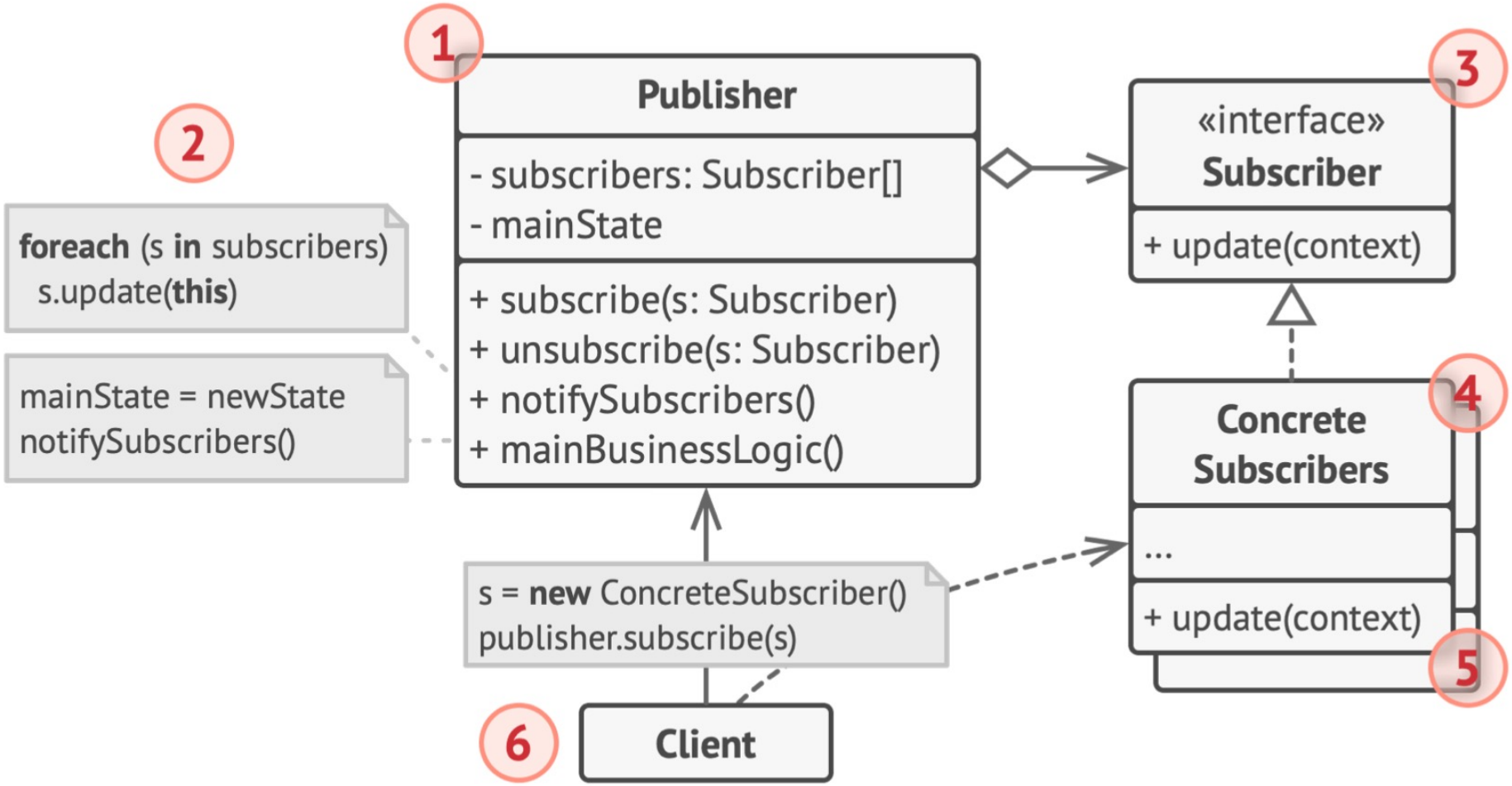
Why?

- You have a resource/object that changes state, and other objects need to be aware of the change as it directly affects them

- Think about event driven interfaces (Graphical User Interfaces)
  - When you interact with the controls on the screen code in the background is waiting to respond to the actions you take
  - They are “subscribed” to that event

- Also common for some networking applications or other devices capable of generating data at unknown intervals

Observer Pattern UML



Pros and Cons

+ *Open/Closed Principle.* You can introduce new subscriber classes without having to change the publisher's code (and vice versa if there's a publisher interface)
+ You can establish relations between objects at runtime

- Subscribers are notified in a random order

Strategy Pattern

Strategy Pattern

- behavioral design pattern
- define a family of algorithms, each in a separate class
- make the algorithms objects interchangeable

Why?

- Many related classes differ only in behavior
- Different variants of an algorithm are needed, often for different space/time tradeoffs
- The algorithm uses data the client should not know about, or had dependencies we want to leave out of the client
- A class has many behaviors, and there are multiple conditional statements in the operations

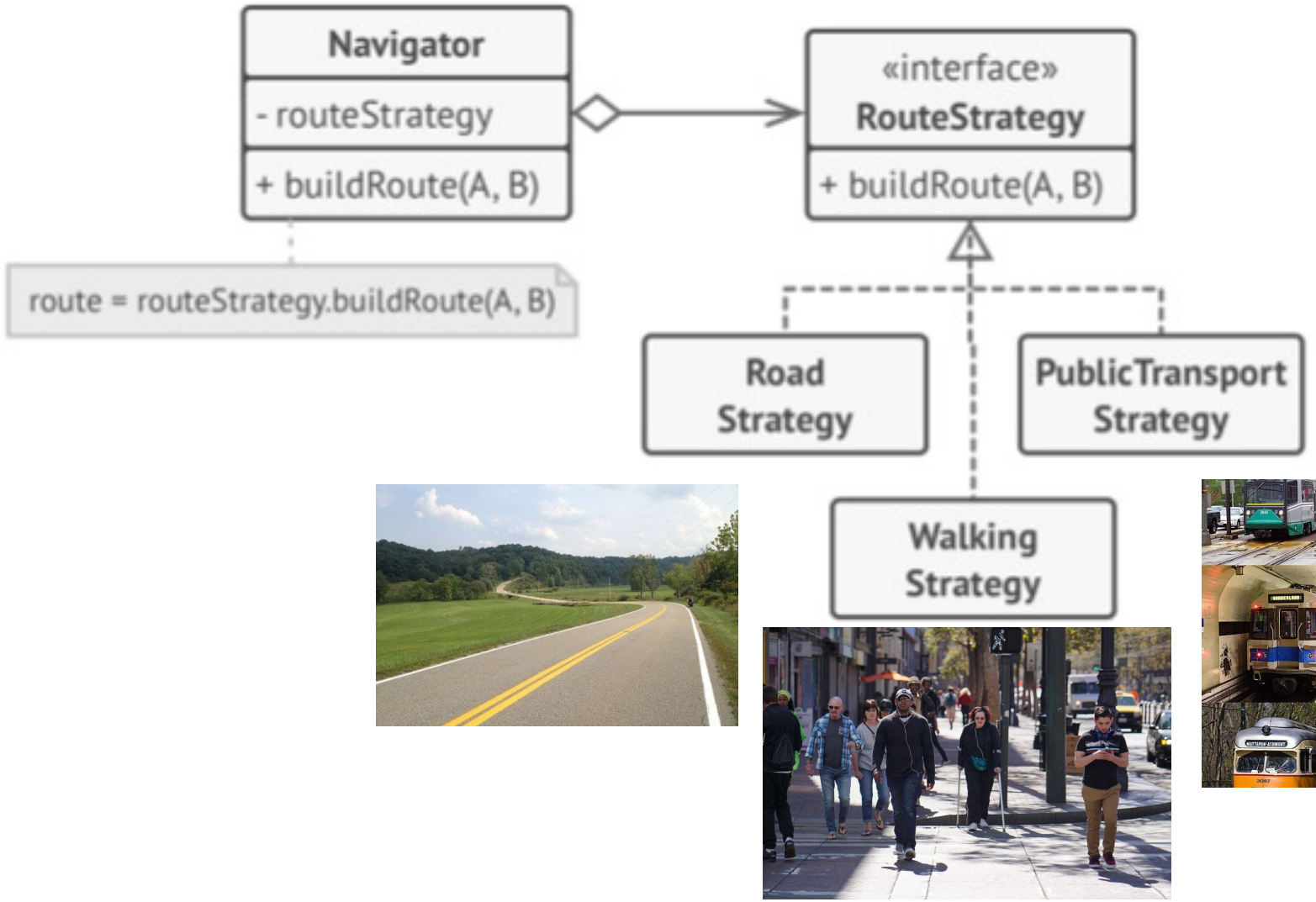
Example

Navigator
...
+ buildRoute(A, B)

How?

- Move all the different algorithms from a class into separate classes called *strategies* that inherit from a class that defines the strategy interface
- We can have our original class, which we call the *context*, store a reference to one of the different strategies.
- In this way, the context delegates to the strategy for algorithmic assistance rather than having that responsibility
- The context does not select the strategy, instead the client passes the desired strategy to the context

Example Revisited



Can expand for biking too!

Pros and Cons

+ You can swap algorithms used inside the object at runtime
+ Implementation details of an algorithm are isolated from the code that uses it
+ You can replace inheritance with composition
+ Satisfies the Open/Closed Principle as you can introduce new strategies without changing the context

- Limited value if you only have a small number of algorithms that rarely change
- Clients must be aware of the differences between the strategies to select the appropriate one
- Can use a set of anonymous functions to accomplish the same thing in supporting languages with less extra classes/interfaces

More resources

- You can find a great catalog of design patterns at the link below.
  - https://refactoring.guru/design-patterns/
  - Examples in multiple languages
  - Good for reference with clear potential use cases, benefits, and drawbacks
  - Basis for the slide material
  - The paid PDF version has essentially the same content as the website
- If you don't mind a Java based focus and would like a friendly read
  - Head First Design Patterns 2nd Edition is available on O'Reilly for free with your Wooster login
- If you'd like to check out the original "Gang of Four" book from '94 can also be found on O'Reilly
  - Design Patterns: Elements of Reusable Object-Oriented Software