Design Patterns

CS6.401 Software Engineering

Dr. Karthik Vaidhyanthan

karthik.vaidhyanathan@iiit.ac.in

https://karthikvaidhyanathan.com

Acknowledgements

The materials used in this presentation have been gathered/adapted/generated from various sources as well as based on my own experiences and knowledge

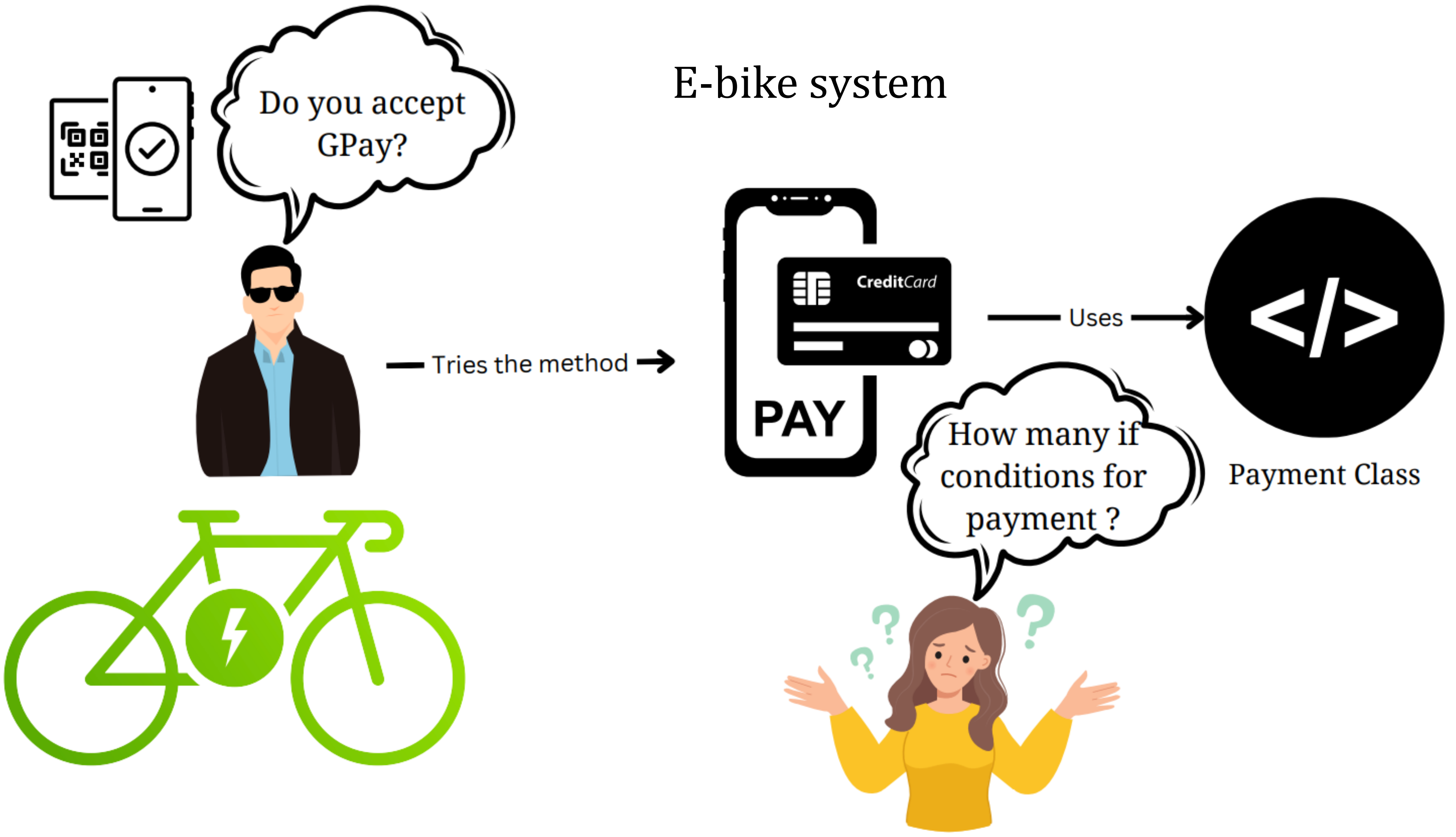
-- Karthik Vaidhyanathan

Sources:

1. **Design Patterns: Elements of Reusable Object-Oriented Software** by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides
2. **Head first Design Patterns**, Second Edition, Eric Freeman and Elisabeth Robson

Strategies can be different: Strategy Pattern! [Behavioral]

Meet the Strategy Pattern!



Meet the Strategy Pattern

- What if you want to alter objects behavior at run-time?
- What if there are similar objects but the way they work is different?
- Each variety of algorithm may require its own set of data and functions

Strategy Pattern: Documentation

Intent

Define a family of algorithms, encapsulate each one and ensure they are interchangeable. Strategy lets algorithm change depending on the client, who is using it

Also Known As: Policy

Motivation

- Different algorithms will be appropriate at different times
- Promotes maintainability
- Two key objects: *Context and Strategy*

Example: Think of Google maps -> selection of mode of transport

Strategy Pattern: Documentation

Applicability

- Many related classes differ only in their behavior
- There is a need for different variants of an algorithm
- Algorithm might require data that client needs not know about – avoid exposing algorithm specific data structures
- Class defines many behaviors and these appear as multiple conditional statements

Strategy Pattern: Documentation

Structure

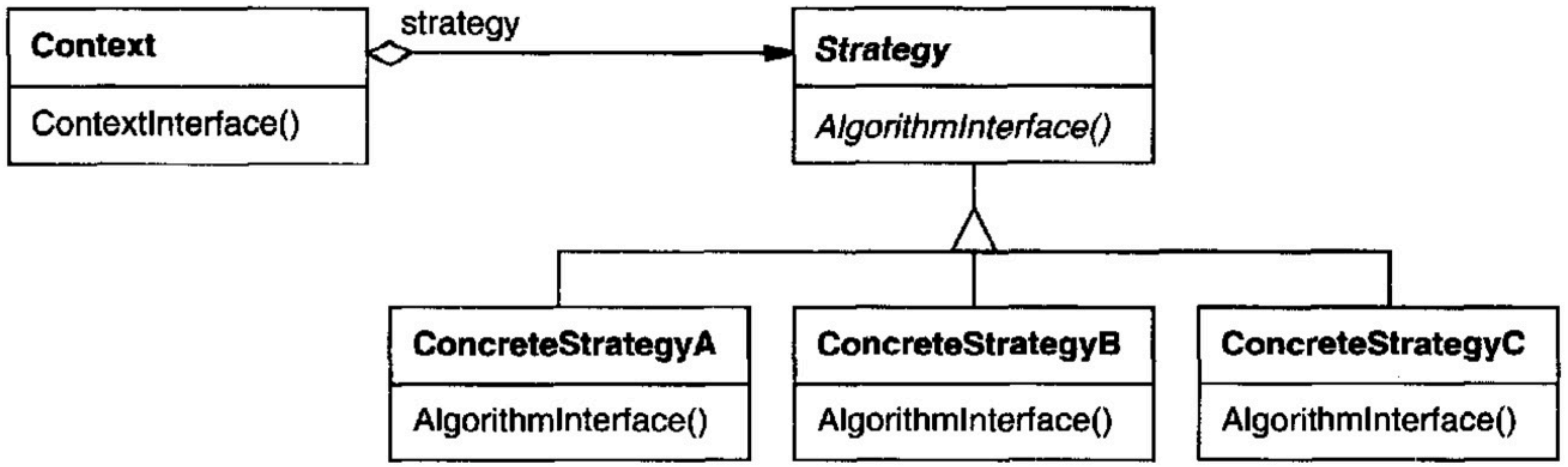


Image source: Gang of four book

Strategy Pattern: Documentation

Participants

Strategy(PaymentType)

- Interface common to all algorithms. Used by context

ConcreteStrategy (DebitCard)

- Implements algorithm using strategy interface

Context (Booking)

- Configured with ConcreteStrategy object
- Maintains reference to a Strategy object
- Can define interface for Strategy to access data

This Photo by Unknown Author is licensed under CC BY

Strategy Pattern: Documentation

Consequences

- Families of related algorithms
  - Hierarchies of strategy classes define a family of algorithms or behaviors
  - Inheritance can help in factoring out common functionality
- Alternative to subclassing
  - Inheritance is another mechanism – Hard-wires context [coupling!]
- Eliminates conditional statements
  - Encapsulates behavior separately [Good solution for long method smell]
- If the number of variations are less - Don't overcomplicate!
- Classes must be aware of different possible strategies

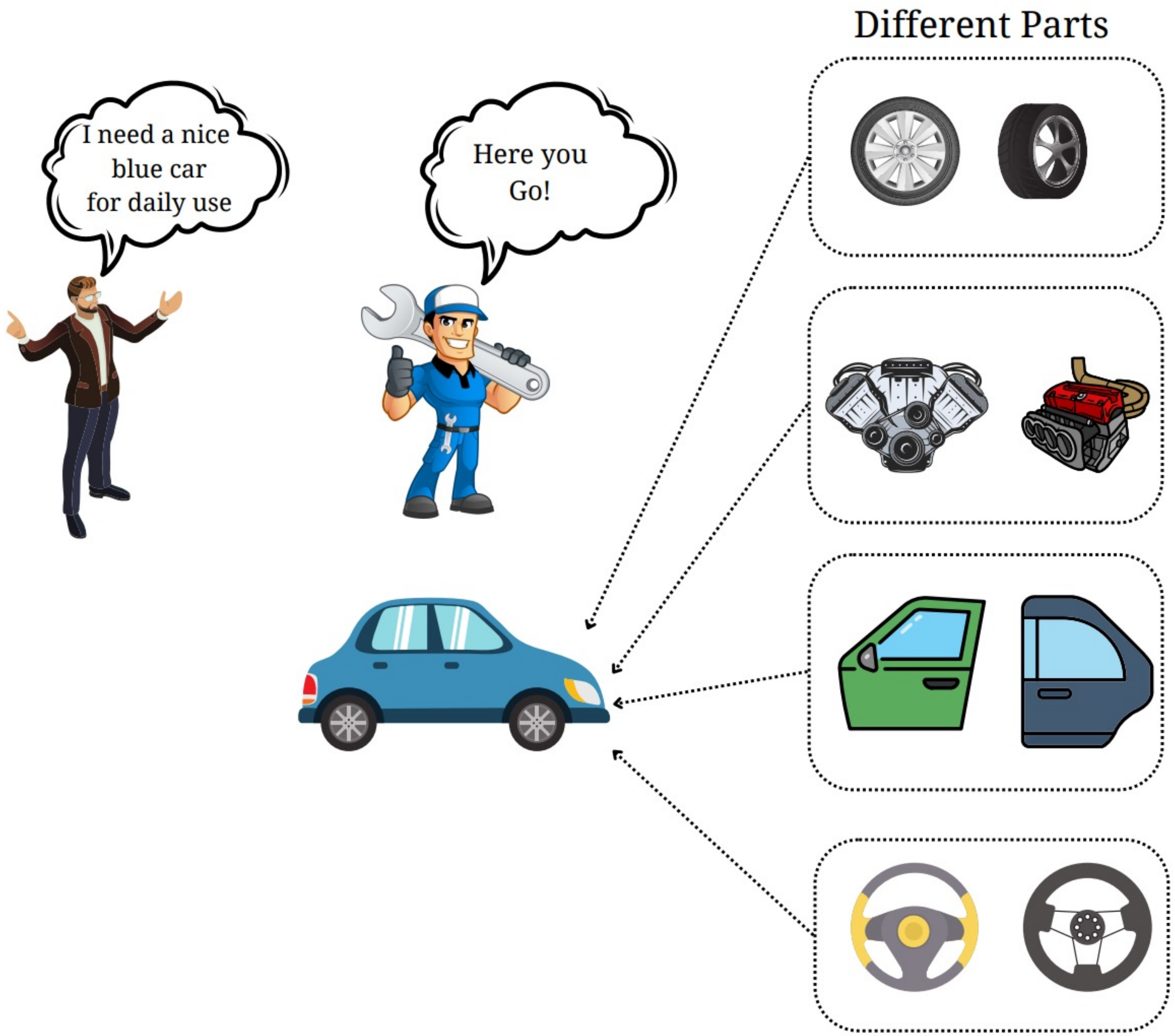
Strategy Pattern: Documentation

Implementation

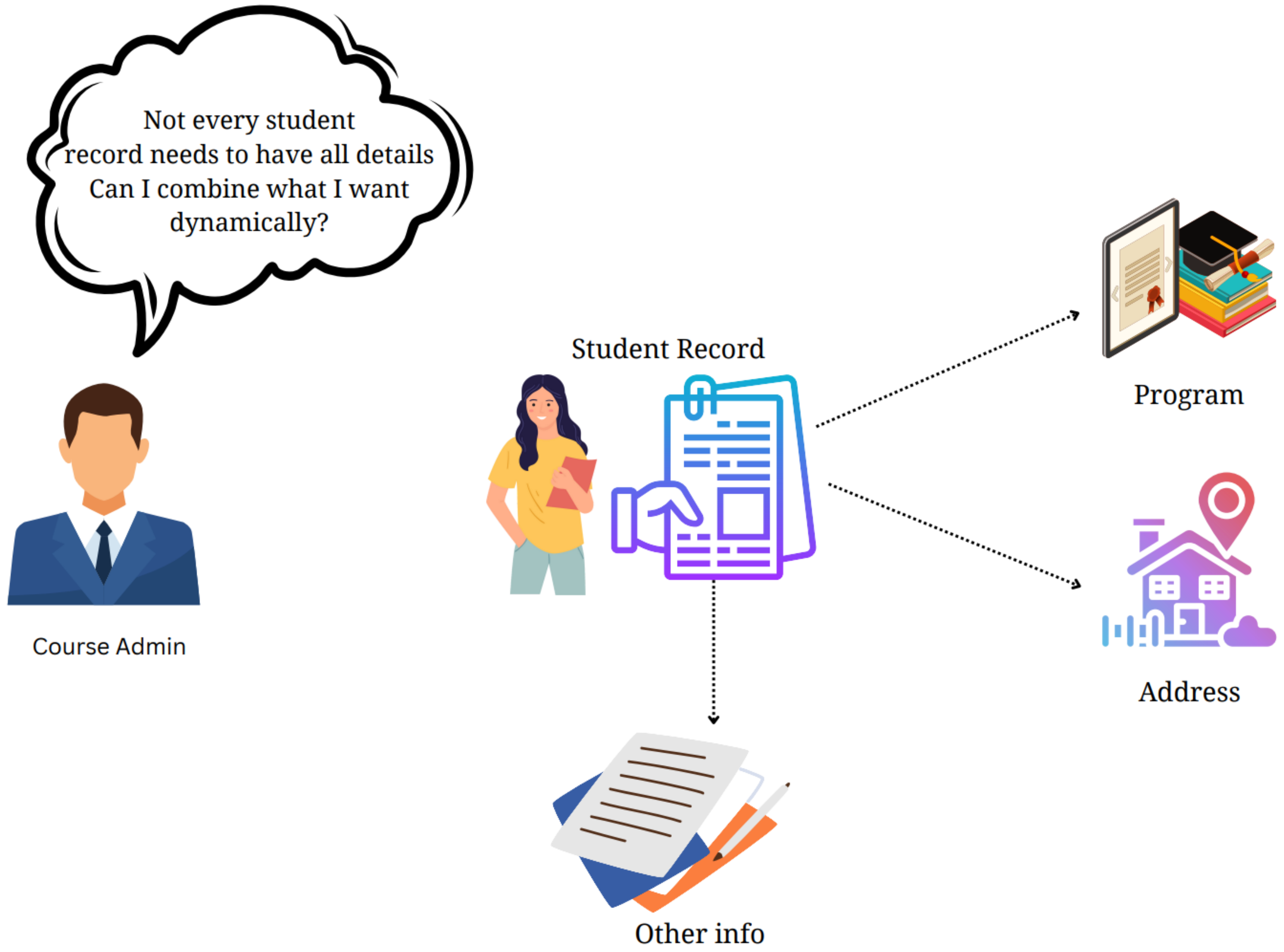
Check the source code given along: EBikePaymentStrategy

How about building things: Builder Pattern! [Creational]

Meet the Builder Pattern!



Meet the Builder Pattern!


Not every student record needs to have all details Can I combine what I want dynamically?
Course Admin
Student Record
Program
Address
Other info

How to dynamically build the different types of student records?

Meet the Builder Pattern

- What if there is a complex object?
- Can we avoid instantiation of a huge constructor?
- Not every time all constructor parameters are required
- Allows extraction of object construction code to separate object
- Creation of an object is just about assembling other objects step by step
- A very decoupled approach to creation

Builder Pattern: Documentation

Intent

Separate construction of complex object from representation such that same construction process can result in different representations

Also Known As: Builder

Motivation

- Separate object construction from business logic
- Promote readability and understandability
- Three key objects: *Director, Builder, Product*

Example: Builder to build different types of vehicles [Each has engine, tyre, etc]

Builder Pattern: Documentation

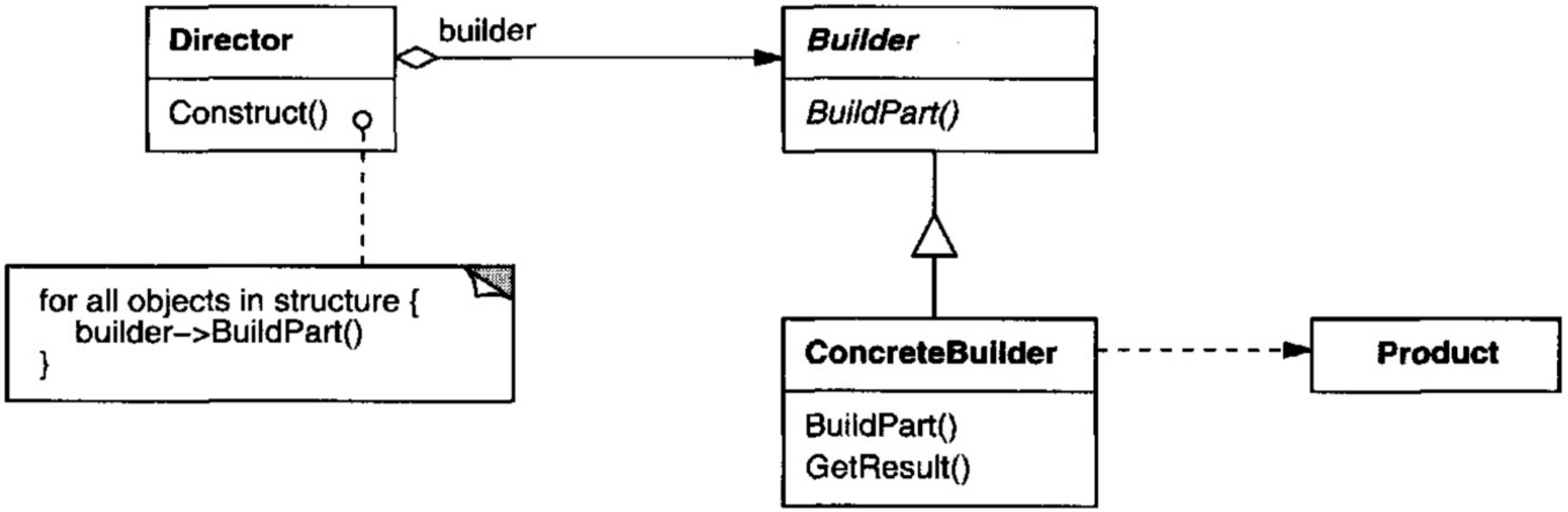
Applicability

- Algorithm for creating the object must be independent
    - Different parts may make up the object
    - Need not worry about how they are put together

- Construction of different representations of the object needs to be supported

Builder Pattern: Documentation

Structure



```
for all objects in structure {
    builder->BuildPart()
}
```

Image source: Gang of four book

Builder Pattern: Documentation

Participants

Builder (StudentBuilder)

- Defines the interface for creating parts of a product object

ConcreteBuilder (ConcreteStudentBuilder)

- Assembles the parts to create product by implementing builder interface

Director (StudentDirector)

- Constructs an object using the builder interface

Product (Student)

- Complex object under construction
- Includes classes that define the different parts

This Photo by Unknown Author is licensed under CC BY

Builder Pattern: Documentation

Consequences

- Easily vary products internal representation
  - Director gets the abstract interface to build a product
  - All that needs to be done is to define a new kind of builder
- Isolate code for representation and constructions
  - Concrete builder contains code for building a kind of product
  - Directors can reuse builders to build different variants of product
- More control over the construction process
  - Step by step approach under directors control – Focus is on the process
- The overall code complexity increases due to multiple classes
  - Benefits in the long run

Builder Pattern: Documentation

Implementation

Check the source code given along: StudentRecordBuilder

Thank You

Course website: karthikv1392.github.io/cs6401_se

Email: karthik.vaidhyanathan@iiit.ac.in
Web: https://karthikvaidhyanathan.com
Twitter: @karthi_ishere

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Software Engineering Research Centre