Creational Pattern: Builder

Kevin Dockx

Architect

@KevinDockx https://www.kevindockx.com

Coming Up

Describing the builder pattern

Structure of the builder pattern

Implementation
- Real-life sample: vehicle builder

Coming Up

Use cases for this pattern

Pattern consequences
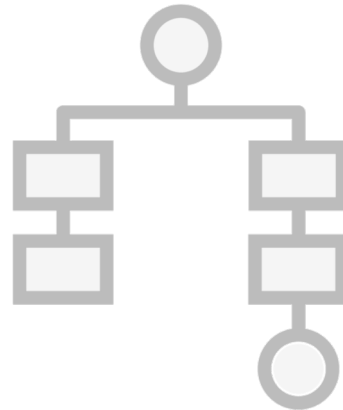
Related patterns
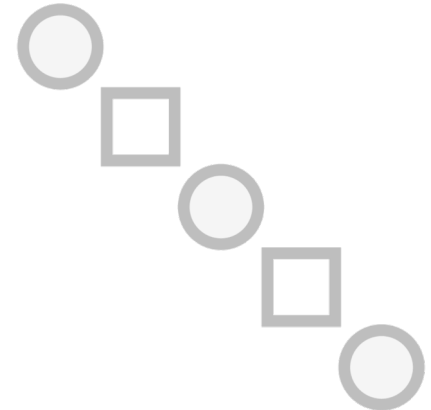
Describing the Builder Pattern

Creational   Structural   Behavioral

Builder

The intent of the builder pattern is to separate the construction of a complex object from its representation. By doing so, the same construction process can create different representations.

Describing the Builder Pattern

Garage

– Needs to construct cars

– Cars are complex and consist of multiple parts (engine, frame, ...)

– Multiple car representations exist (BMW, Mini, ...)

Describing the Builder Pattern

CarBuilder
BuildFrame()
BuildEngine()
Car Car

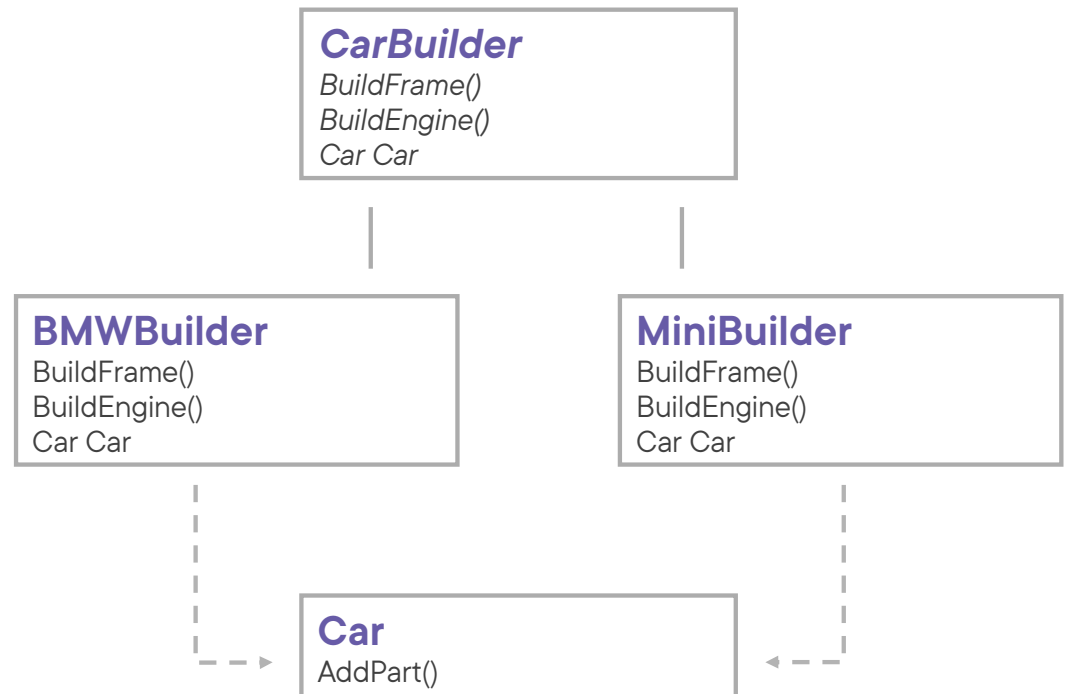# Describing the Builder Pattern

CarBuilder
BuildFrame()
BuildEngine()
Car Car

Car
AddPart()

Describing the Builder Pattern

CarBuilder *(italic)*
BuildFrame()
BuildEngine()
Car Car

BMWBuilder
BuildFrame()
BuildEngine()
Car Car

MiniBuilder
BuildFrame()
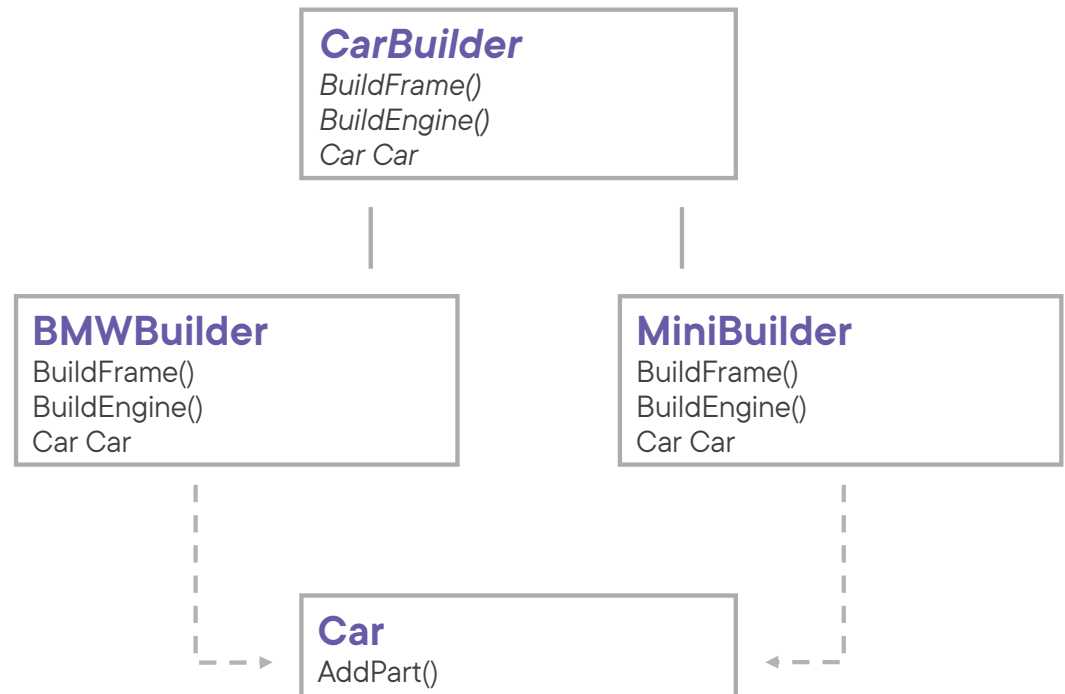BuildEngine()
Car Car

Car
AddPart()

Creating a complex car object is transparent for the consumer of the builder

Work on interfaces, not implementations
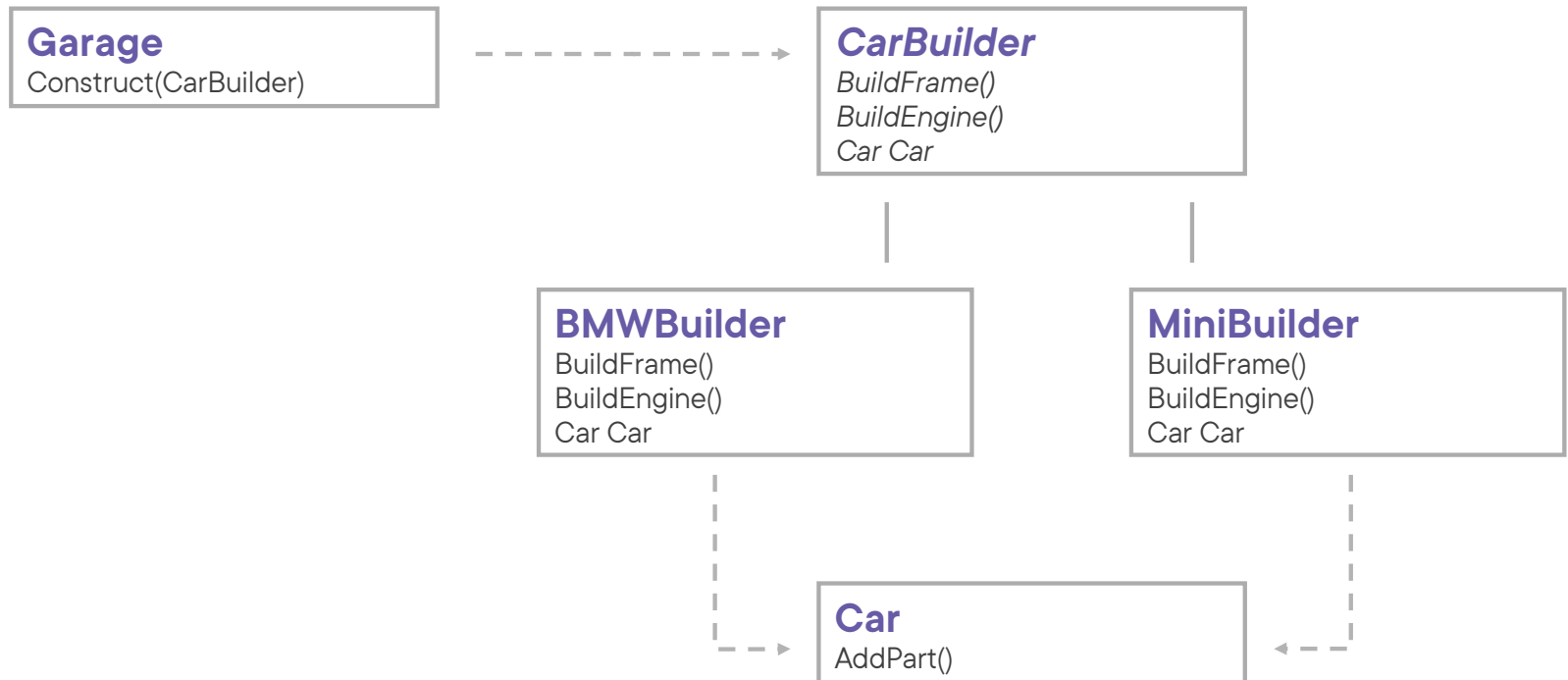
Describing the Builder Pattern

CarBuilder *(italic)*
BuildFrame()
BuildEngine()
Car Car

BMWBuilder
BuildFrame()
BuildEngine()
Car Car

MiniBuilder
BuildFrame()
BuildEngine()
Car Car

Car
AddPart()

Describing the Builder Pattern

Garage
Construct(CarBuilder)

CarBuilder
BuildFrame()
BuildEngine()
Car Car

BMWBuilder
BuildFrame()
BuildEngine()
Car Car

MiniBuilder
BuildFrame()
BuildEngine()
Car Car

Car
AddPart()

Builder Pattern Structure

Garage
Construct(CarBuilder)

CarBuilder
BuildFrame()
BuildEngine()
Car Car

BMWBuilder
BuildFrame()
BuildEngine()
Car Car

MiniBuilder
BuildFrame()
BuildEngine()
Car Car

Car
AddPart()

Builder specifies an abstract interface for creating parts of a **Product** object

Builder Pattern Structure

Garage
Construct(CarBuilder)

- - - → *CarBuilder*
BuildFrame()
BuildEngine()
Car Car

BMWBuilder
BuildFrame()
BuildEngine()
Car Car

MiniBuilder
BuildFrame()
BuildEngine()
Car Car

Car
AddPart()

Builder Pattern Structure

Garage
Construct(CarBuilder)

- - - - - - - → ***Builder***
BuildSomePart()
BuildAnotherPart()
Product GetResult()

BMWBuilder
BuildFrame()
BuildEngine()
Car Car

MiniBuilder
BuildFrame()
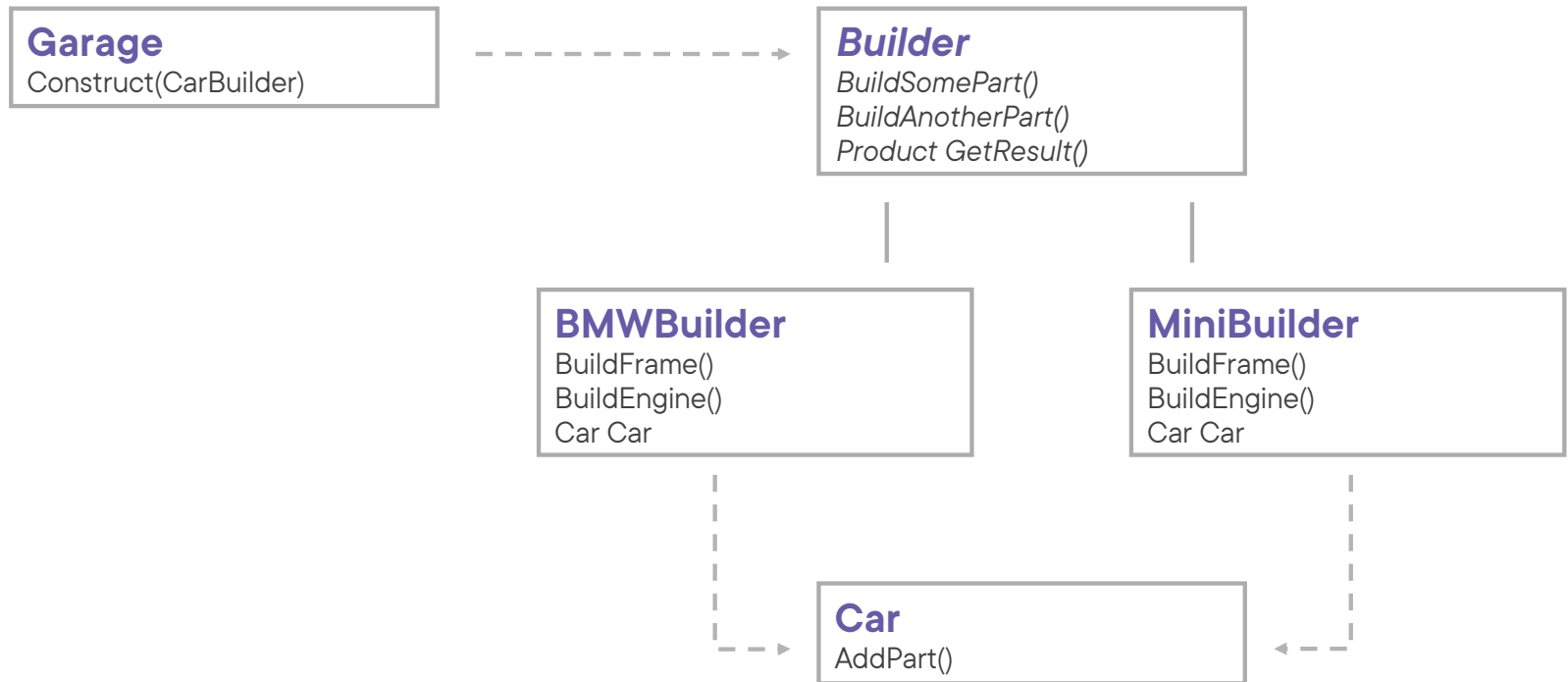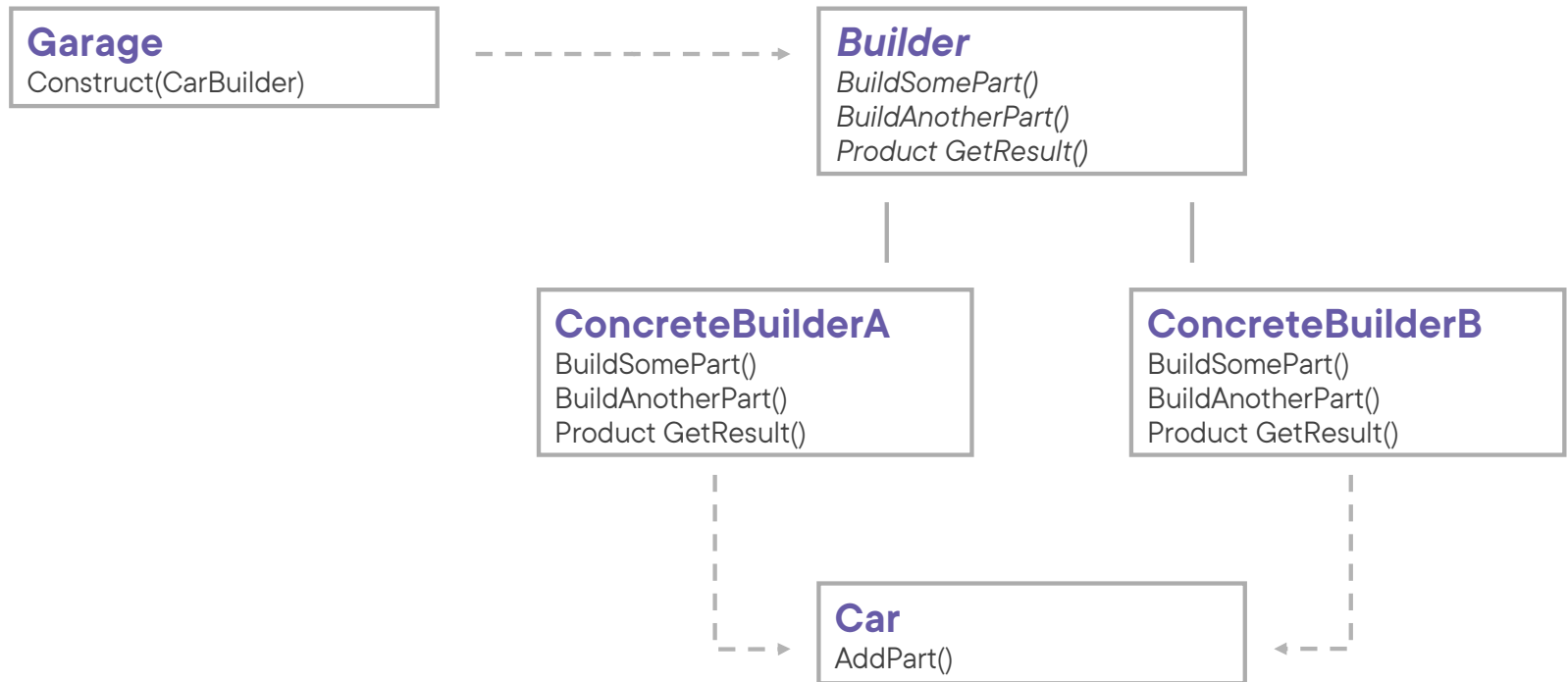BuildEngine()
Car Car

Car
AddPart()
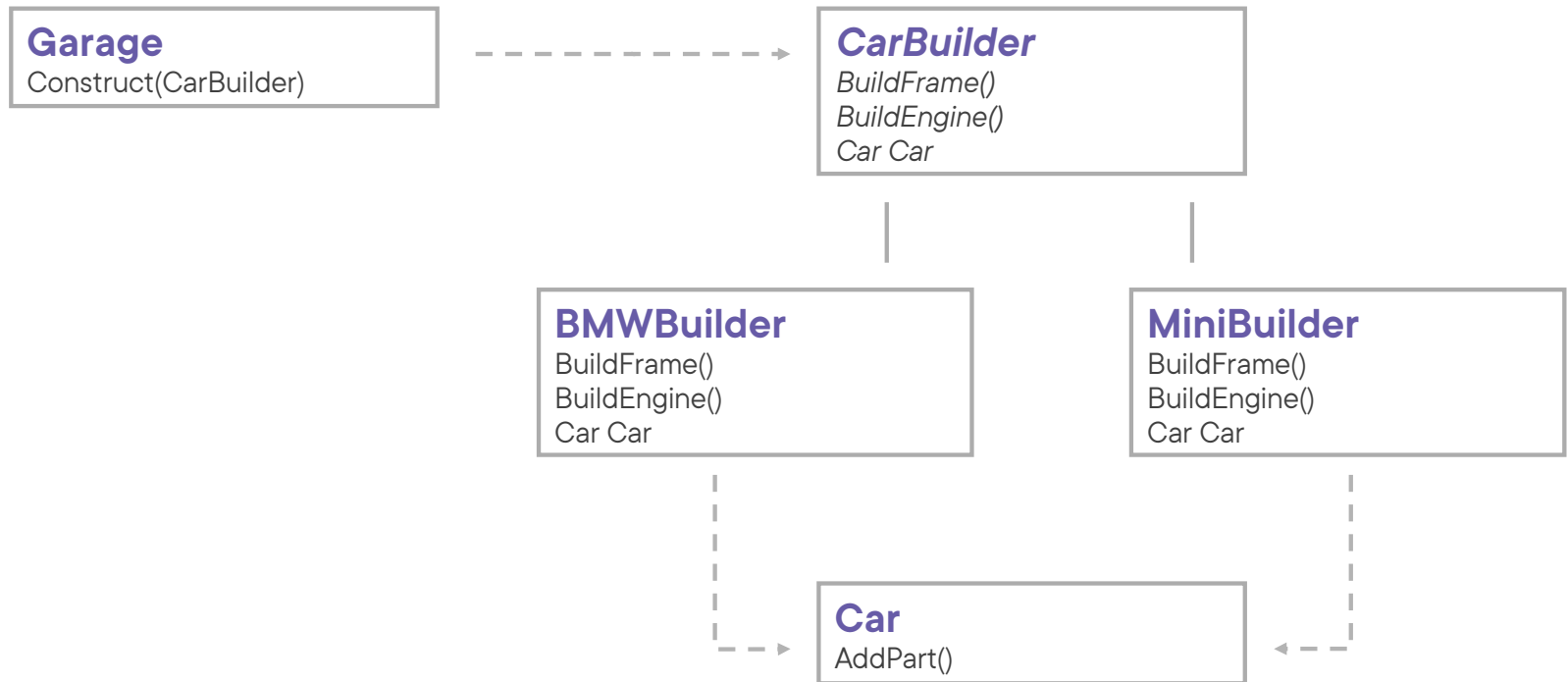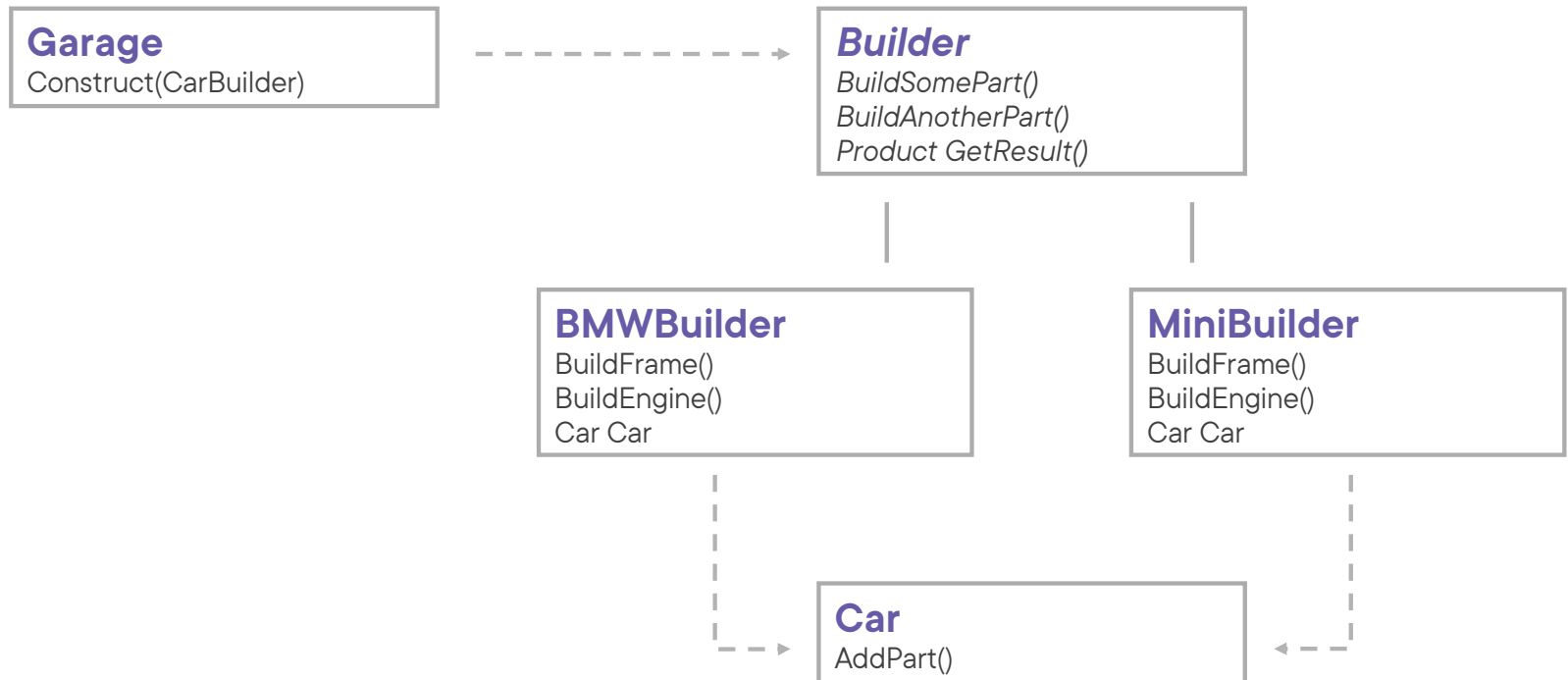
ConcreteBuilder constructs and assembles parts of the **Product** by implementing the **Builder** interface. It keeps track of the representation it creates, and provides an interface for retrieving the **Product**.

Builder Pattern Structure

Garage
Construct(CarBuilder)
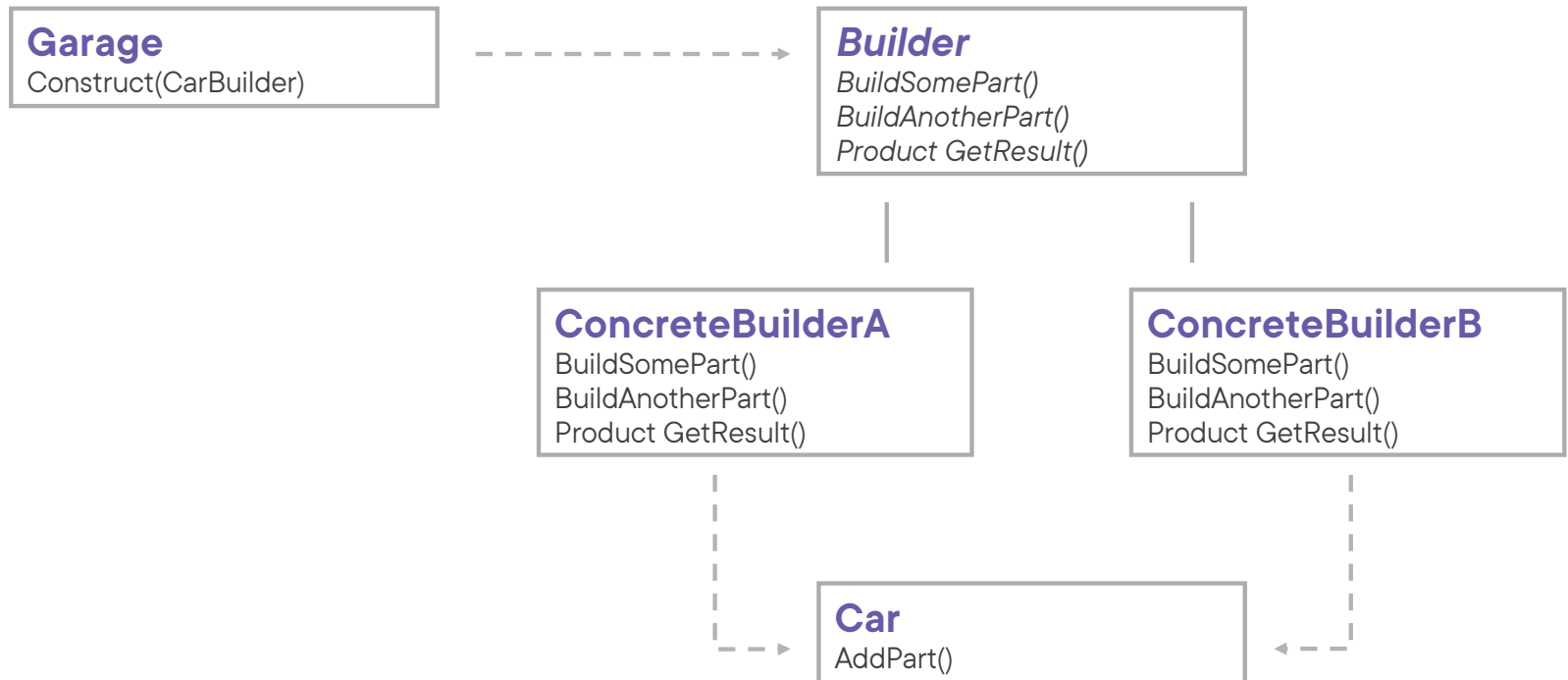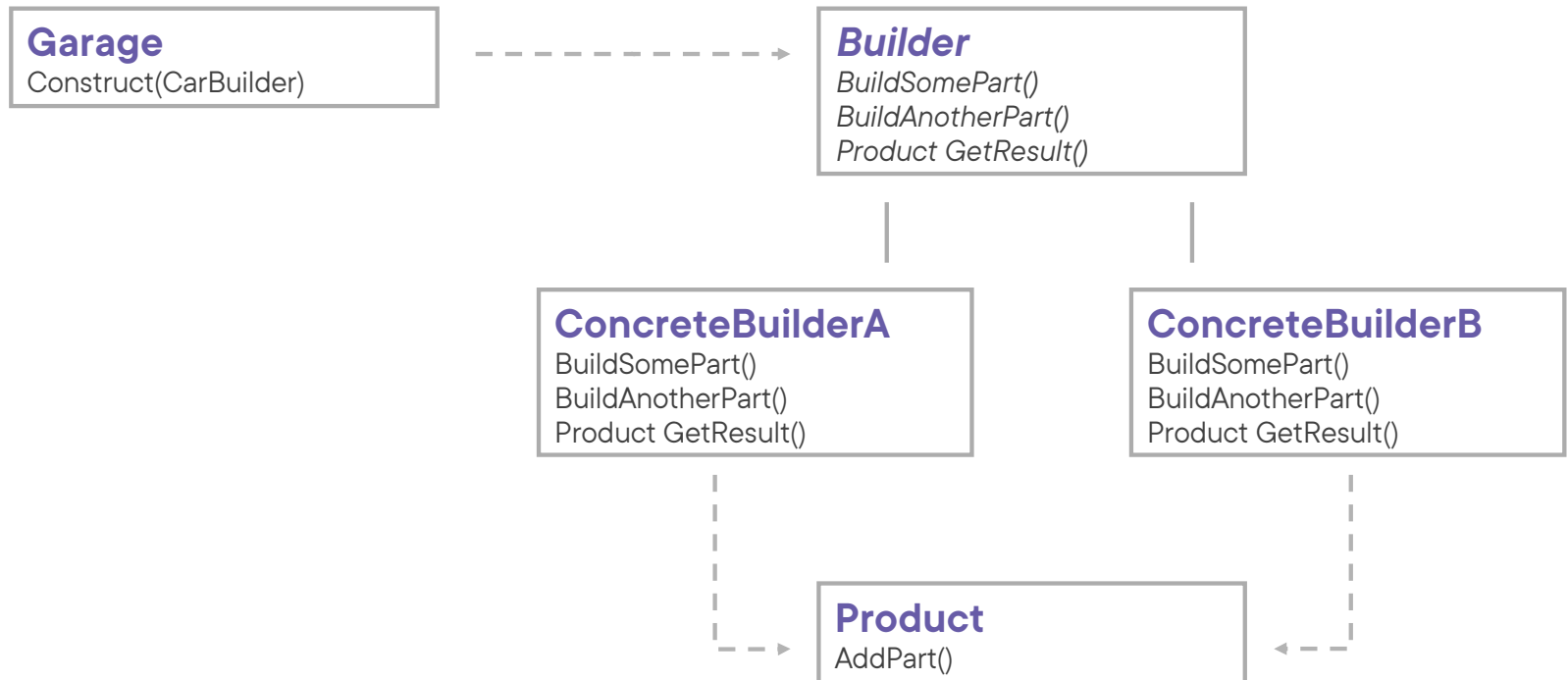
- - - - - - - - ->

Builder
BuildSomePart()
BuildAnotherPart()
Product GetResult()

BMWBuilder
BuildFrame()
BuildEngine()
Car Car

MiniBuilder
BuildFrame()
BuildEngine()
Car Car

Car
AddPart()

Builder Pattern Structure

Garage
Construct(CarBuilder)

- - - - - - - - - - ▸

Builder
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderA
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderB
BuildSomePart()
BuildAnotherPart()
Product GetResult()

Car
AddPart()

Product represents the complex object under construction

Builder Pattern Structure

Garage
Construct(CarBuilder)

Builder
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderA
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderB
BuildSomePart()
BuildAnotherPart()
Product GetResult()

Car
AddPart()

Builder Pattern Structure

Garage
Construct(CarBuilder)

Builder
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderA
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderB
BuildSomePart()
BuildAnotherPart()
Product GetResult()

Product
AddPart()

Director constructs an object by using the **Builder** interface

Builder Pattern Structure

Garage
Construct(CarBuilder)

Builder
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderA
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderB
BuildSomePart()
BuildAnotherPart()
Product GetResult()

Product
AddPart()

Builder Pattern Structure

Director
Construct()

- - - - - - - - - ->

Builder
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderA
BuildSomePart()
BuildAnotherPart()
Product GetResult()

ConcreteBuilderB
BuildSomePart()
BuildAnotherPart()
Product GetResult()

Product
AddPart()

Demo

Implementing the builder pattern

Use Cases for the Builder Pattern

When you want to make the algorithm for creating a complex object independent of the parts that make up the object and how they're assembled

When you want to construction process to allow different representations for the object that's constructed
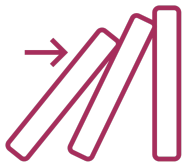
Pattern Consequences

It lets us vary a products' internal representation

It isolates code for construction and representation; it thus improves modularity by encapsulating the way a complex object is constructed and represented: single responsibility principle
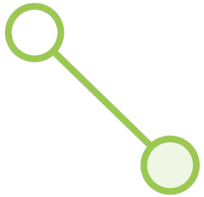
It gives us finer control over the construction process

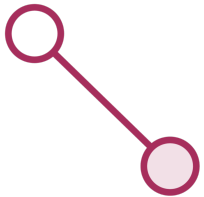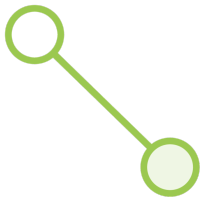**Complexity of your code base increases**

Related Patterns

Abstract factory
Both can be used to construct complex objects, but the builder
constructs the complex objects step by step

Singleton
A builder can be implemented as a singleton

Composite
Composites are often built by builders

Summary

Intent of the builder pattern:

- Separate the construction of a complex object from its representation, so the same construction process can create different representations

Use it when you want to make the algorithm for creating a complex object independent of the parts that make up the object and how they're assembled

Summary

Implementation:

- Define an abstract base class or interface as `Builder`
- Have the director work on the `Builder`, not on `ConcreteBuilder` implementations

Up Next:
Creational Pattern: Prototype