Strategy Pattern

Strategy Pattern Problem

Sometimes in your application you have similar objects with different behaviors and you want to have flexibility to manage the behaviors.

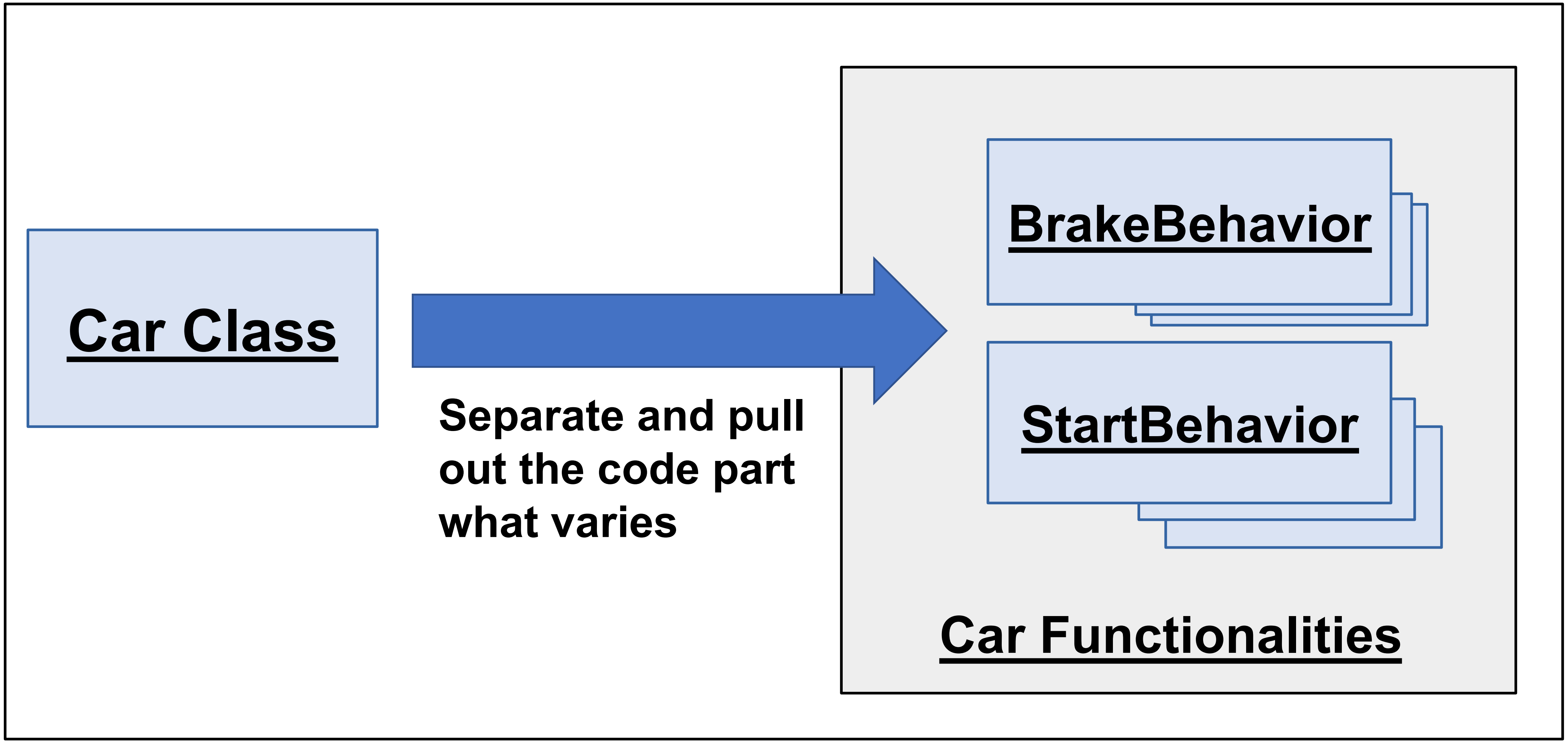
Your software should better select the behavior (or computations) dynamically at run time.

Strategy Pattern – Naïve Solution

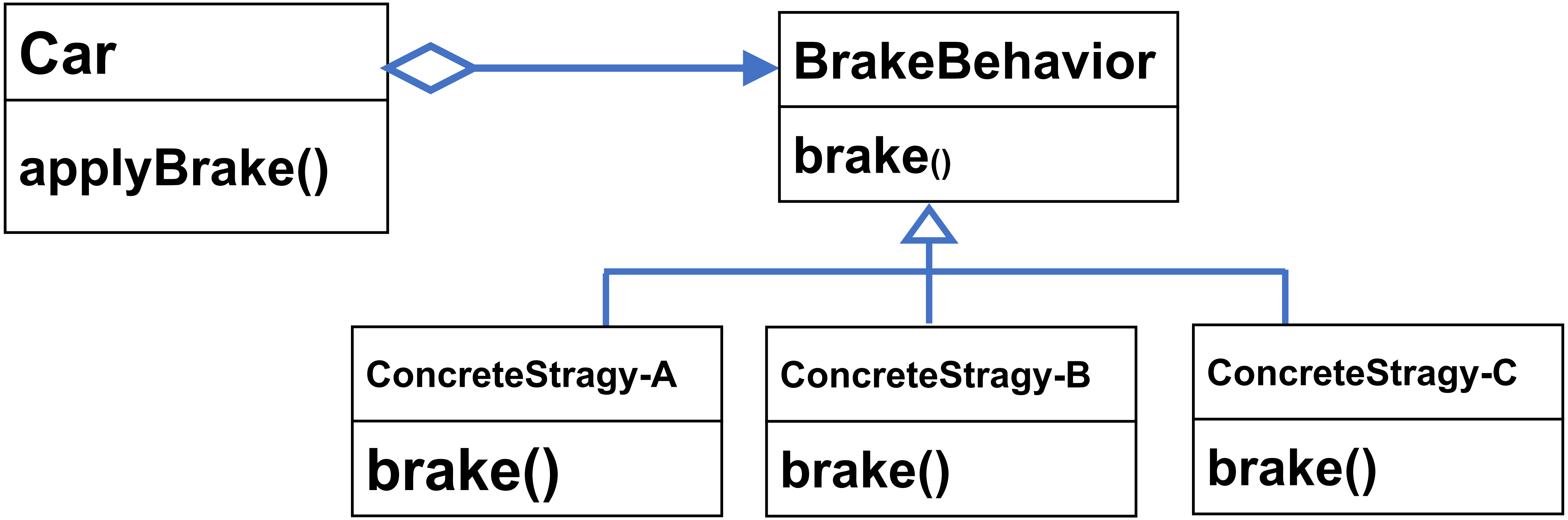
We can hard-code different behaviors and implement them in a long “IF THEN ELSE” statement or SWITCH statement.

- The problem is that adding or removing new behaviors requires changing the IF or SWITCH statement.
- Instead if IF statement we can use the strategy pattern.

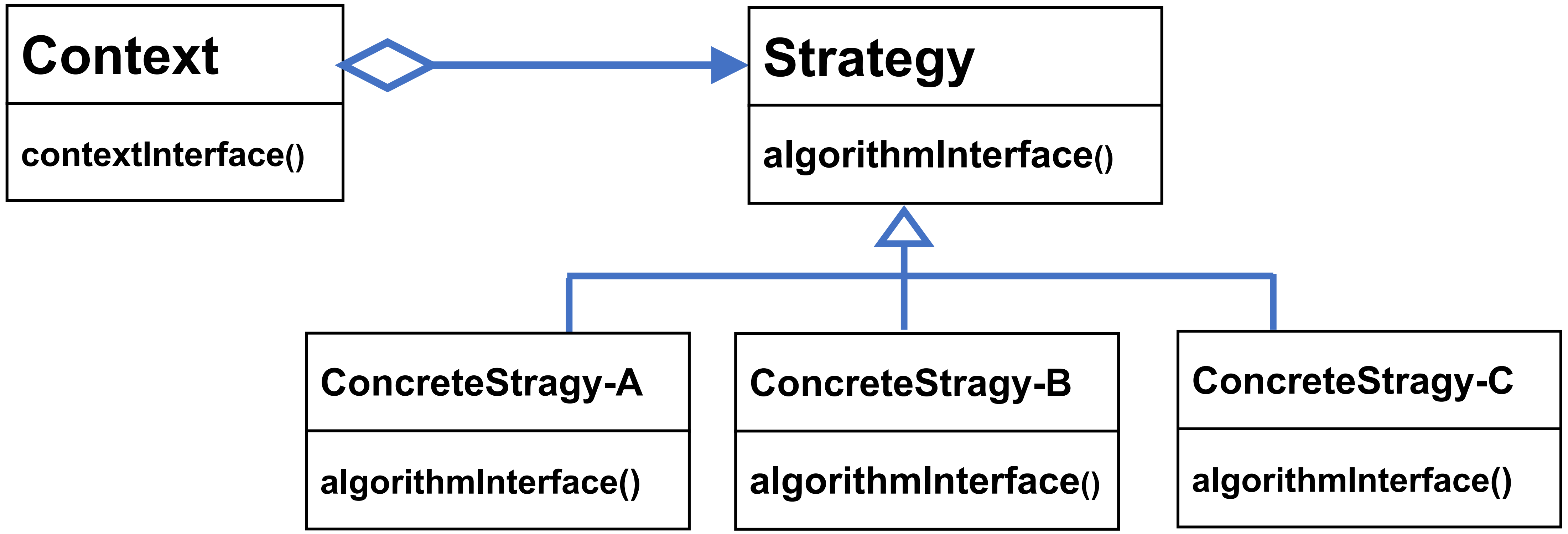
Example of Different Functionalities



Structure of Strategy Pattern



Structure of Strategy Pattern



By using strategy pattern, you can define a family of algorithms/behaviors, encapsulate each one, and make them interchangeable.

Participants in Pattern

- **Strategy**

It declares **an interface** common to all underlaying supported algorithms.

- **Concrete Strategy**

implements the algorithm using the **Strategy interface**

- **Context (Composition)**

It is configured with a **ConcreteStrategy** object and maintains a reference to a Strategy object.

When use the Strategy Pattern

Use the Strategy pattern when:

- You have many related classes **differ only in their behavior**.
- You have **different variants of an algorithm or functionality**.
- You want to **avoid exposing complex, algorithm-specific data structures to clients**. An algorithm uses data that you want to hide it from client.
- You have **a class that defines many behaviors** and these have multiple conditional statements in its operations.

Benefits of Using Strategy Pattern

+ Defines a family of related algorithms/behaviors to **reuse**.

+ An **alternative to sub-classing and inheritance**

+ Strategies eliminate conditional statements like having switch cases inside the algorithm implementation. **The case statement will be eliminated** by delegating the task selection to a Strategy object.

+ Strategies can expose to client **different implementations of the same behavior.**

Drawbacks of Using Strategy Pattern

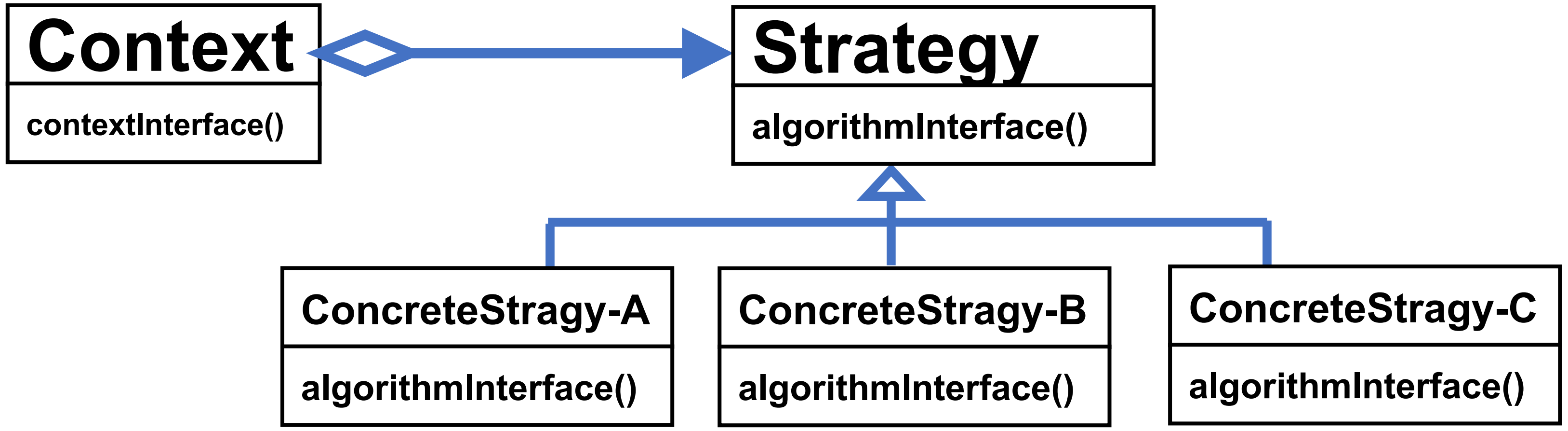
- **Clients must be aware** of different Strategies. Use strategy only when the variation in behavior is relevant to clients.

- **Communication overhead** between Strategy and Context. Interfaces are shared among all strategies, info is passed through interfaces.

- **Increased number of objects**

Strategy Pattern - Implementation

- You need a class that uses the algorithm/context contain an abstract class (strategy) including an abstract method to call the algorithm.
- Then each derived class implements its own algorithm.
- **Note:** the method is abstract because you wanted to have different behavior.
- **Note:** the responsibility for selecting the particular implementation is done by the client object and passed to the context.



Strategy Pattern

The Strategy Pattern defines a family of algorithms or behaviors, encapsulates and separate each one, and then makes them interchangeable, and expose them via interfaces.