DefineX

Introduction to Design Patterns

Structural Design Patterns

by Adem Aldemir

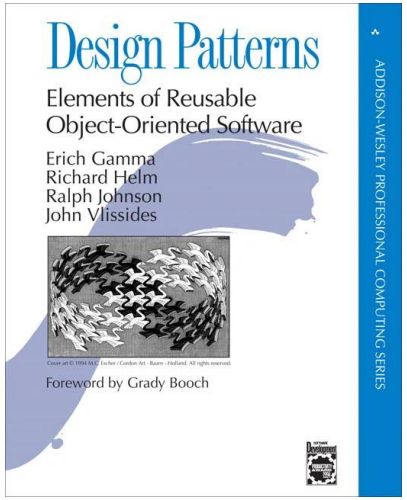
Design Patterns

What is Design Patterns?

First referenced in early 80's, with the emerge of OOP. Formally defined in 1994 in the GOF book (even before Java!). The Design Pattern Idea originated from Christopher Wolfgang Alexander. "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." says *Christopher Wolfgang Alexander*. Even though he was talking about patterns in building and towns, what he says is true about *object-oriented design patterns. Solutions are expressed in terms of object and interfaces instead of walls and doors.*. The recurring aspects of designs are called **design patterns**. A pattern is the outline of the reusable solution to a general problem encountered in a particular context. Many of them have been systematically documented for all software developers to use.

Gang of Four

Design patterns gained popularity in computer science after the book **"Design Patterns: Elements of Reusable Object-Oriented Software"** was published in 1994 by the so -called "Gang of Four" (GoF). GoF established 23 patterns classified into 3 types - Creational, Structural and Behavioral.



Title : *Design Patterns: Elements of Reusable Object-Oriented Software*

Is the first and essential book on patterns and Patterns are well classified and described

Types of Design Patterns

Creational patterns

These patterns provide various object creation mechanisms, which increase flexibility and reuse of existing code.

Structural patterns

These patterns explain how to assemble objects and classes into larger structures while keeping these structures flexible and efficient.

Behavioral patterns

These patterns are concerned with algorithms and the assignment of responsibilities between objects.

Structural Design Patterns

Structural Design Patterns

Structural Patterns

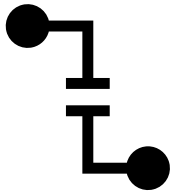
- Ease the design
- How to assemble objects and classes into larger structures
- Keep the structures flexible and efficient

- Bridge
- Composite
- Adapter
- Decorator
- Facade
- Flyweight
- Proxy

Bridge

"Decouple an abstraction from its implementation so that the two can vary independently."

How Bridge works?

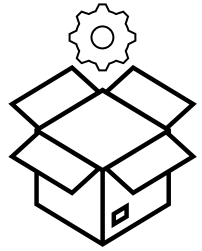


Divides and Organizes

Divides and organizes a single class that has multiple variants of some functionality into two hierarchies : abstractions and implementations.

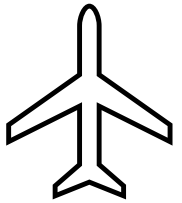
Not Expose Details

Client code won't be exposed to implementation details as it will work only with high level abstractions.



Prefer Composition

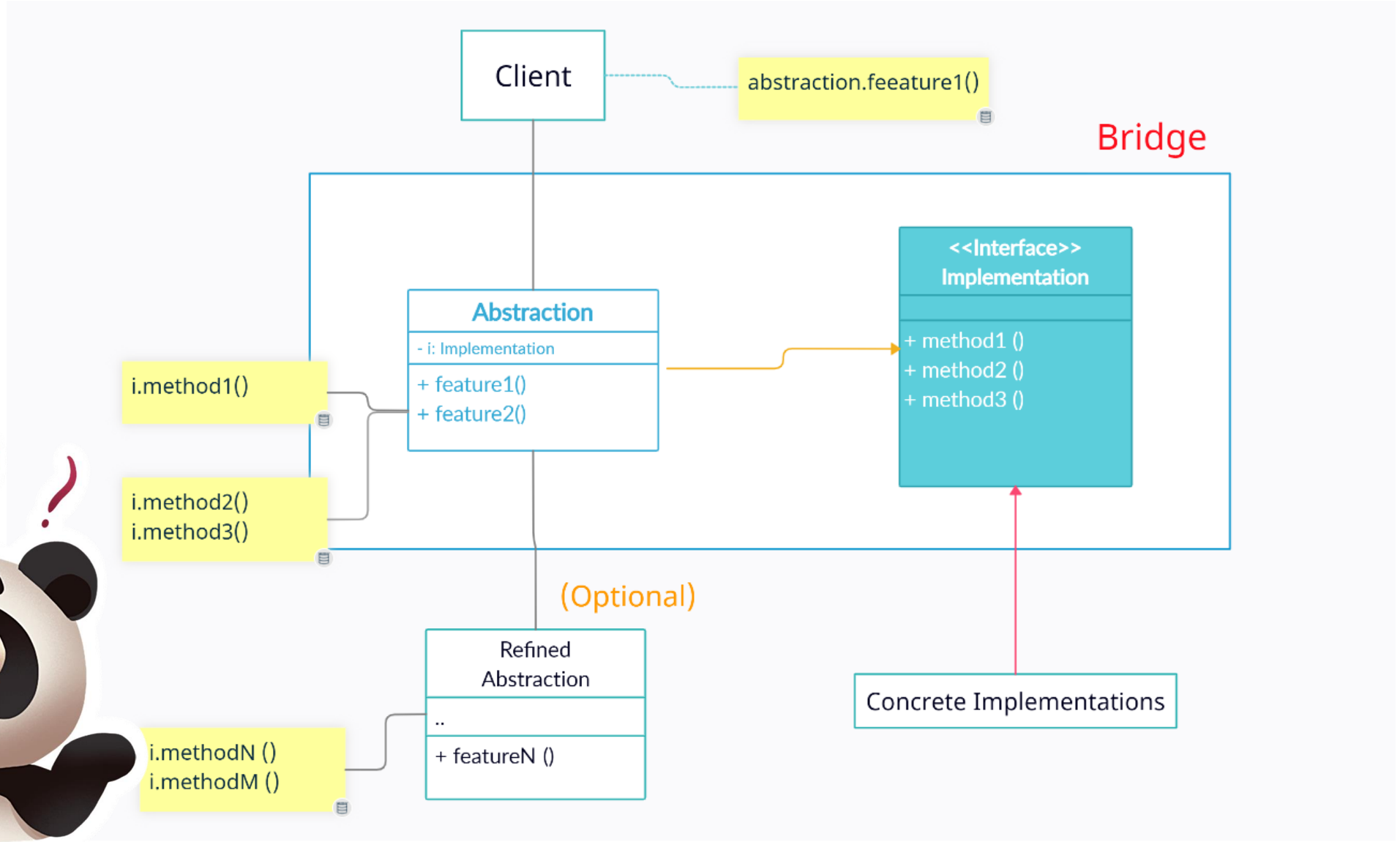
The Bridge pattern is an application of the old advice, “prefer composition over inheritance”.



Be Independent

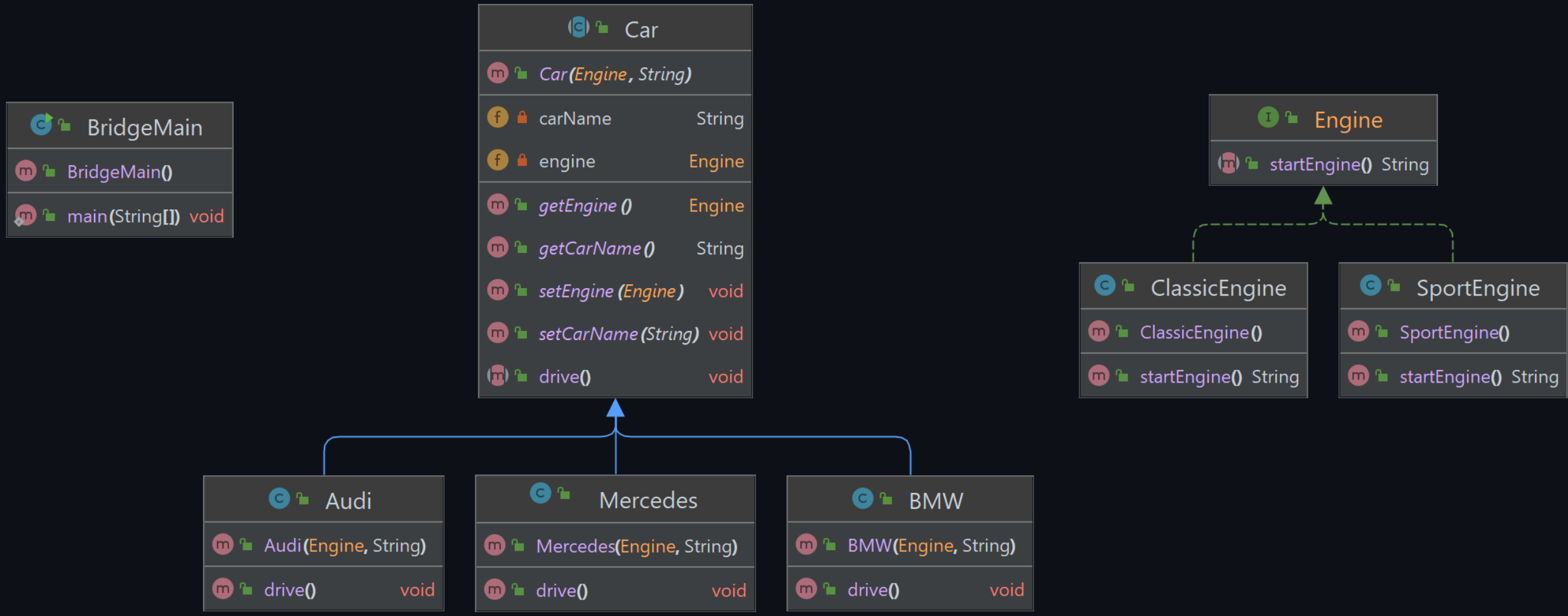
Independently introduce new abstractions and implementations: making it very easy to switch between them at runtime.

Structure of Bridge

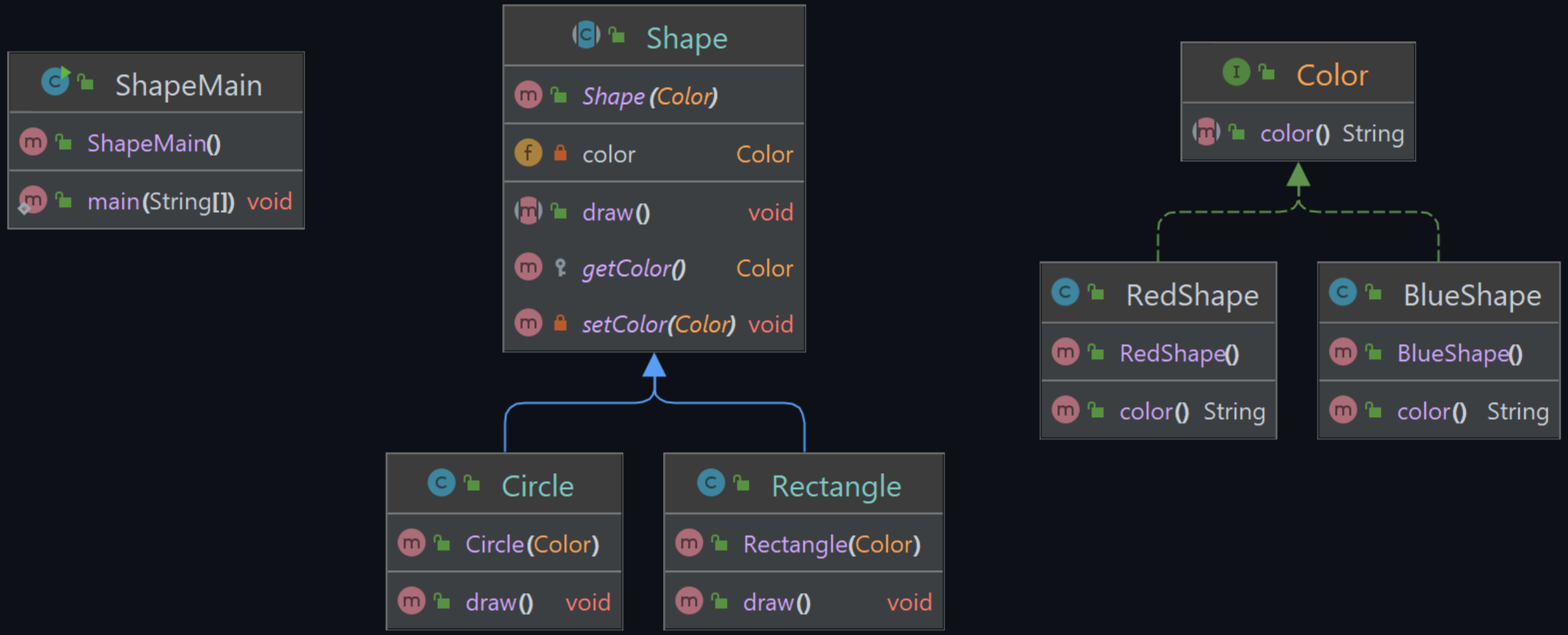


Let's code Bridge.

Car Engine Example



Shape with Color Example



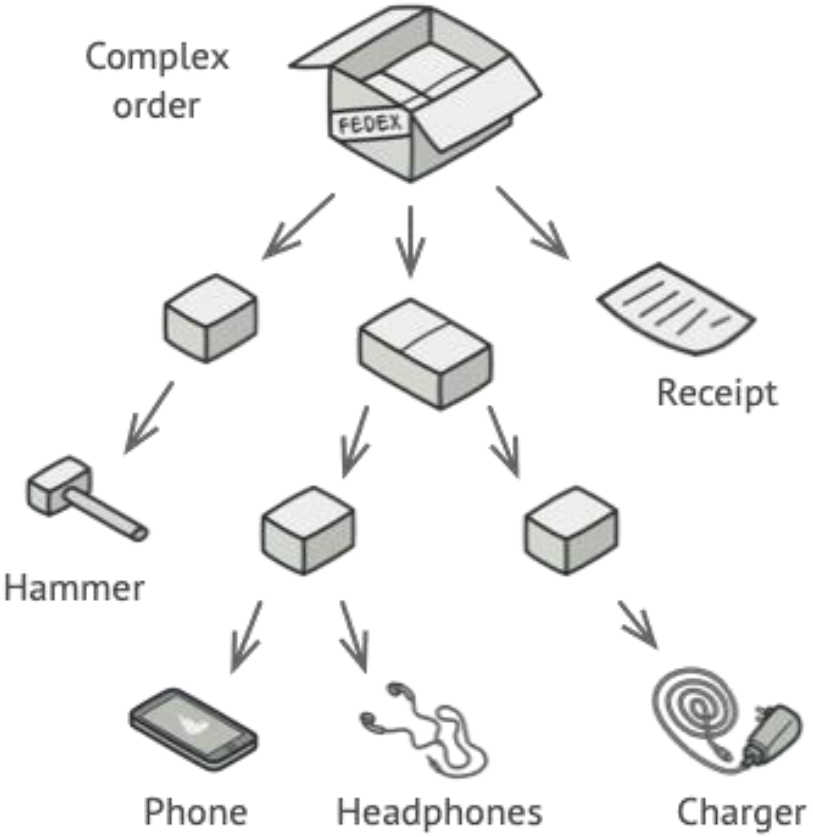
Composite

The Composite design pattern composes objects into tree structures to represent part-whole hierarchies. This pattern lets clients treat individual objects and compositions of objects uniformly.

Composite

Composite is a structural design pattern that lets you compose objects into tree structures and then work with these structures as if they were individual objects.

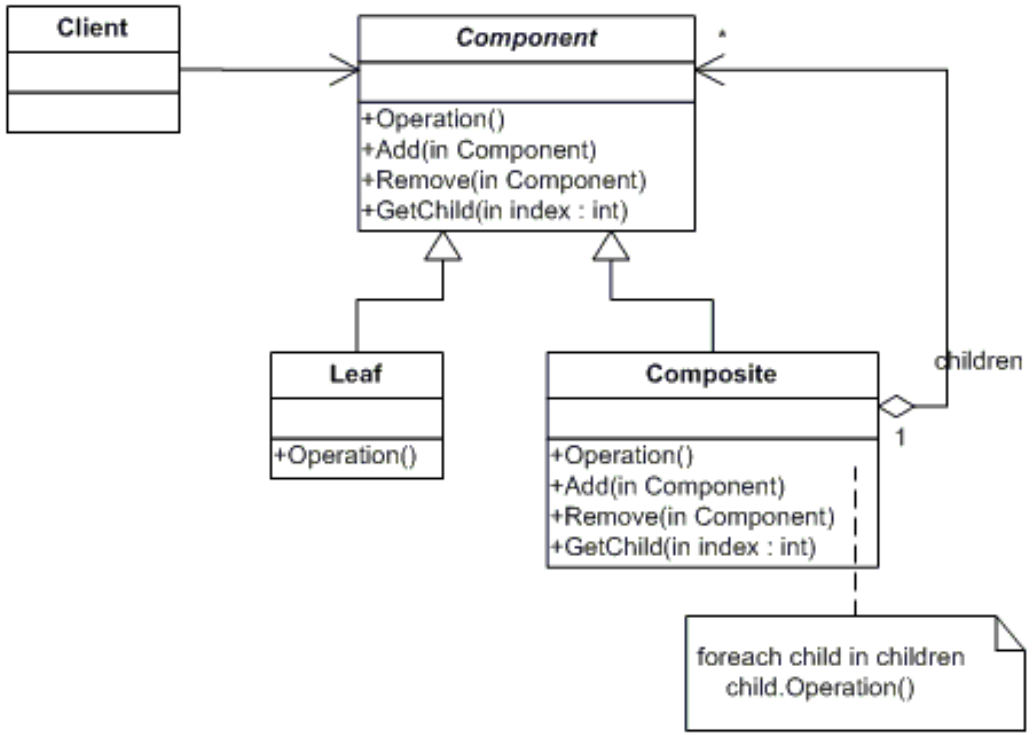
Composite pattern is used where we need to treat a group of objects in similar way as a single object.

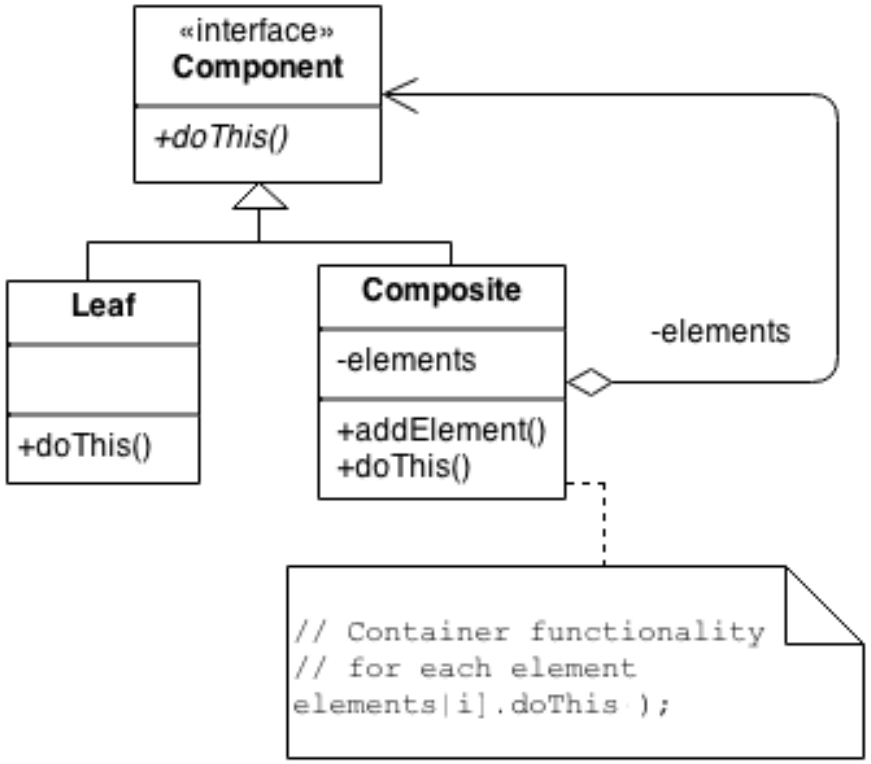


Structure of Composite

1. The **Component** interface describes operations that are common to both simple and complex elements of the tree.
2. The **Leaf** is a basic element of a tree that doesn't have sub-elements.
   Usually, leaf components end up doing most of the real work, since they don't have anyone to delegate the work to.
3. The **Container** (aka **composite**) is an element that has sub-elements: leaves or other containers. A container doesn't know the concrete classes of its children. It works with all sub-elements only via the component interface.
   Upon receiving a request, a container delegates the work to its sub-elements, processes intermediate results and then returns the final result to the client.
4. The **Client** works with all elements through the component interface. As a result, the client can work in the same way with both simple or complex elements of the tree.

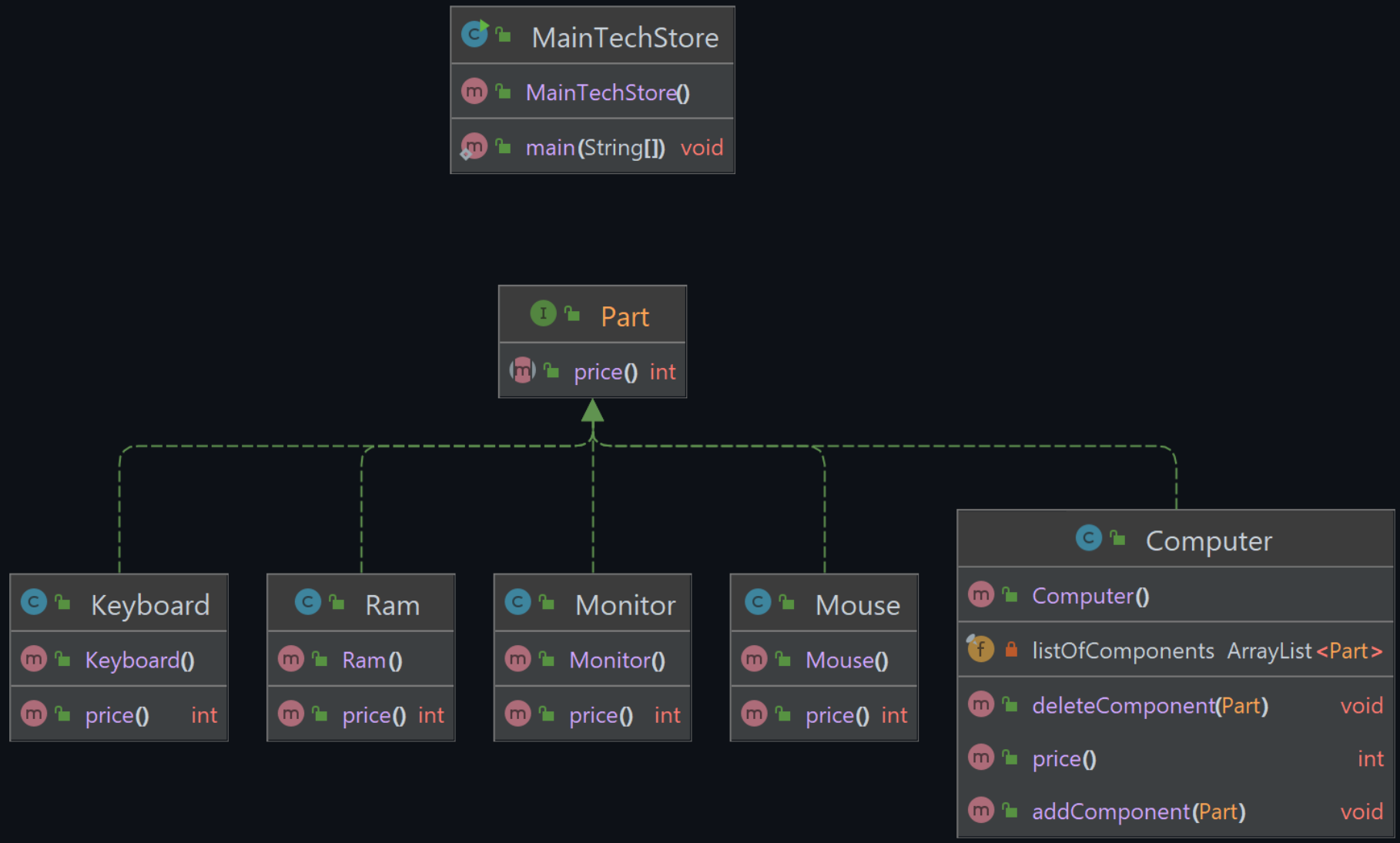
Structure of Composite



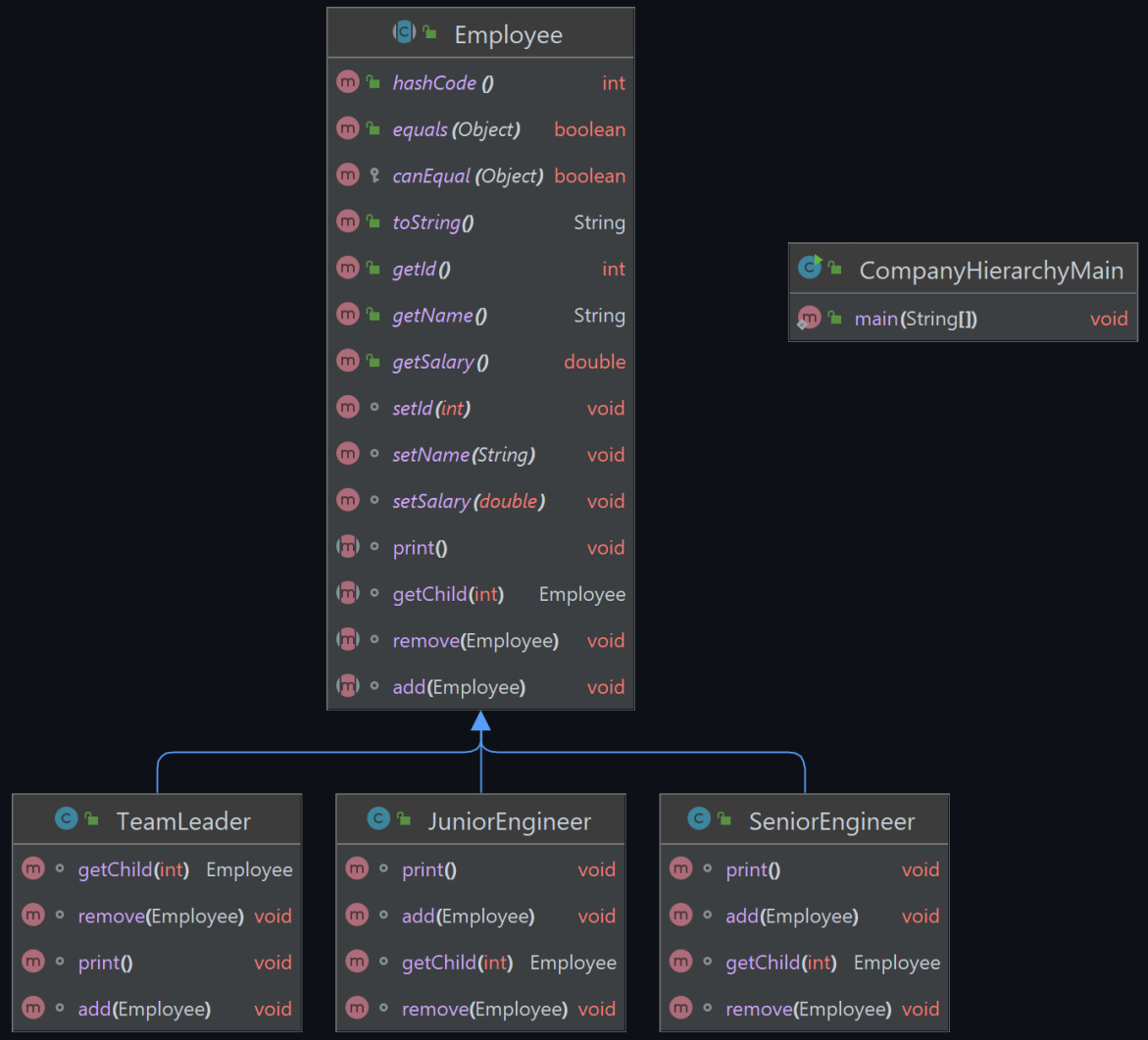


Let's code Composite.

Tech Store Example



Company Hierarchy Example



DefineX

And Thank you X.

Follow me via the Channels.

Sources

https://www.javacodegeeks.com/2015/09/bridge-design-pattern.html
https://www.turkayurkmez.com/bridge-design-pattern/
https://howtodoinjava.com/design-patterns/structural/bridge-design-pattern/
https://springframework.guru/gang-of-four-design-patterns/bridge-pattern/
https://github.com/yusufyilmazfr/tasarim-desenleri-turkce-kaynak#-bridge