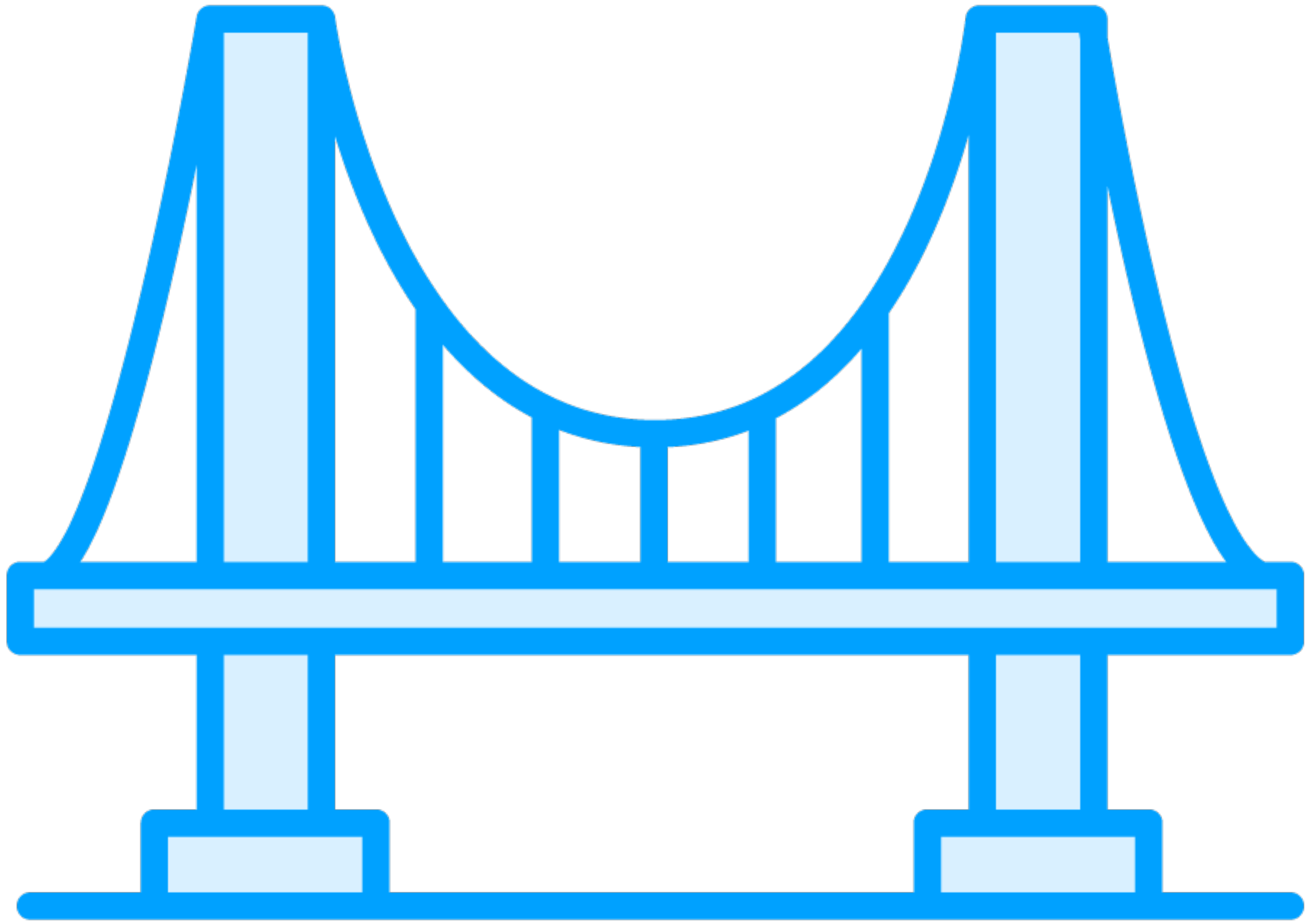
Bridge Design Pattern

Bryan Hansen

@bh5k

Bridge



Concepts

Decouple Abstraction and Implementation

Encapsulation, Composition, Inheritance

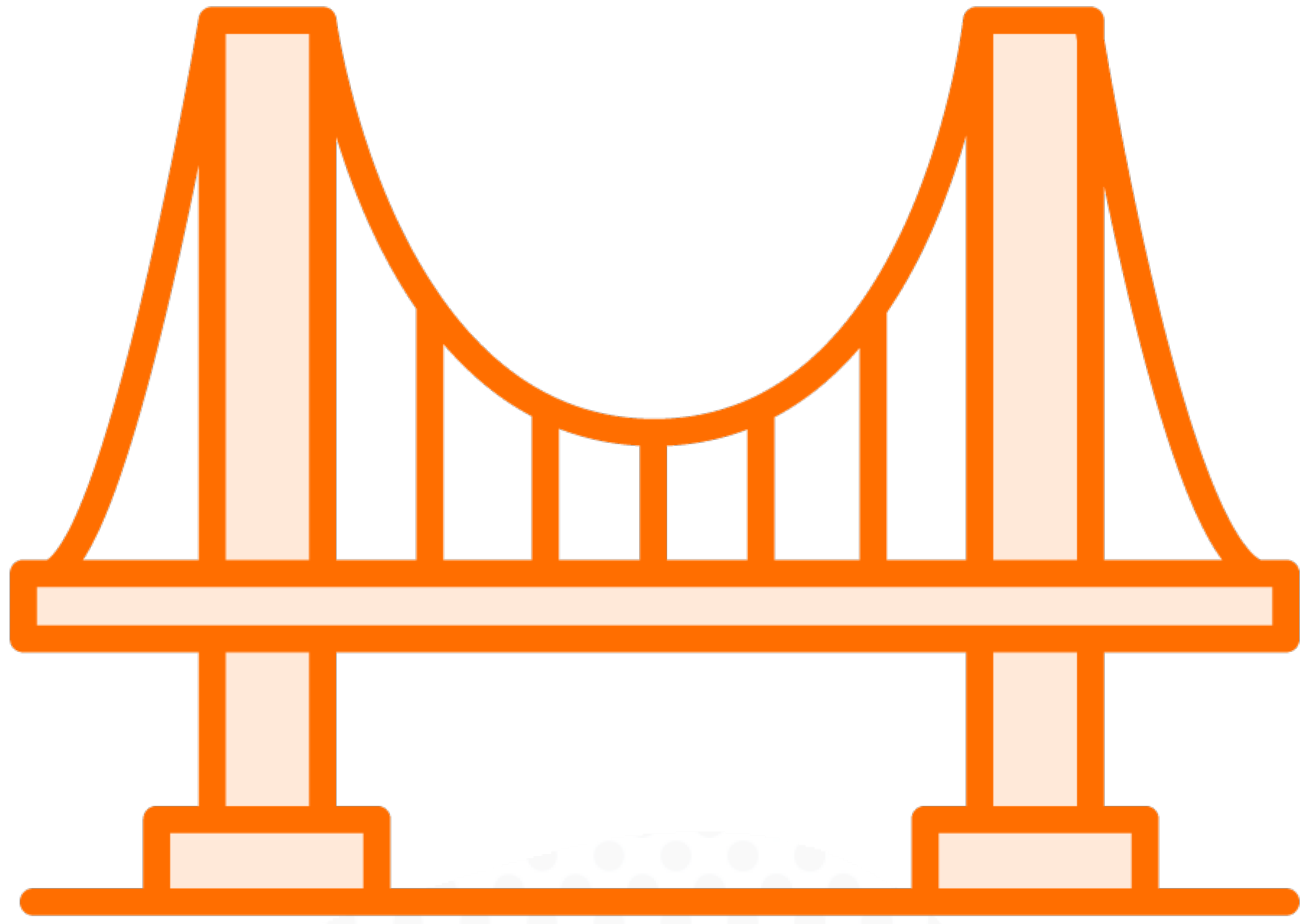
Changes in Abstraction won't affect client

Details won't be right

Examples:

Driver

JDBC



Design

Interfaces and Abstract classes

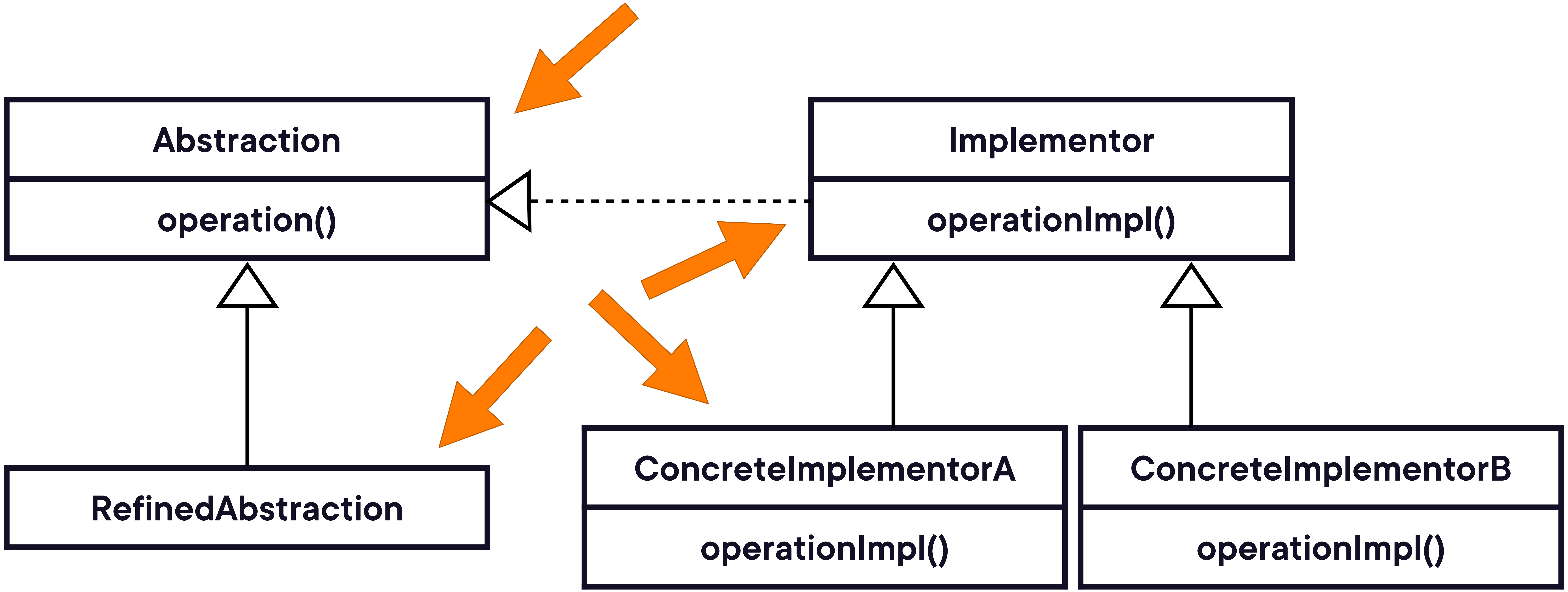
Composition over Inheritance

More than Composition

Expect change from both sides

Abstraction, Implementor, Refined Abstraction, Concrete Implementor

UML


Abstraction
operation()
RefinedAbstraction
Implementor
operationImpl()
ConcreteImplementorA
operationImpl()
ConcreteImplementorB
operationImpl()

Everyday Example - JDBC

```
DriverManager.registerDriver(new org.apache.derby.jdbc.EmbeddedDriver());

String dbUrl = “jdbc:derby:memory:codejava/webdb;create=true";

Connection conn = DriverManager.getConnection(dbUrl);

Statement sta = conn.createStatement();
```

Exercise Bridge

Color and Shape

Color and Shape Bridge

Create Bridge

Another Bridge

Pitfalls

Increases Complexity

Conceptually difficult to plan

More than just OO

What goes where

Contrast

Bridge	VS	Adapter
Designed upfront		Works after code is designed
Abstraction / implementation vary		Legacy
Built in advance		Retrofitted
Both adapt multiple systems		Provides different interface

Bridge Summary

Design for uncertainty

Can be complex

Provides flexibility

More than composition