Design Patterns

Alex Poling

The What

Design patterns are solutions to common problems we find or run into in code

Patterns are about reusable designs that solve problems with cleaner code

From the 23 Gang of Four (GoF) patterns

Based on the book “Design Patterns: Elements of Reusable Object-Oriented Software”

The Why

Proven solutions - no need to reinvent

Language Agnostic

Communication tool

Standards for developers

Three Types

Behavioral - Used to manage algorithms, relationships, and responsibilities. How they behave

Structural - Used to define how objects and classes can be combined

Creational - Create objects to be decoupled from implemented classes

Chain of Responsibilities

Allows multiple objects to be able to handle a request

Which class handles that request is based on what is in that request

DEMO

Command

Encapsulating a request allowing it to be treated as an object

Ex: Calculator app

ICommand

AddCommand implements ICommand - receives the invoke and calls the receiver

Invoker - User

Receiver - Calculator.cs - knows how to perform the action (add) based on the command it receives

Interpreter

The Context - Data being interpreted

The Expression - An abstract class that defines methods needed for conversions

The Terminal- Provides the conversion of different types of data

Uses Java Reflection

Example - Text based game, “Ask a question” app, Gallons to Quarts would be Terminal

Iterator

Allows for access to the elements of an aggregate object without allowing access to its underlying representation

Used for extracting objects from a collection without exposing the collection itself

DEMO (depending on time)

Mediator

Memento

DEMO

Memento.cs contains state of an object to be restored

Originator.cs creates and stores states in Memento.cs

Caretaker is responsible to restore object state from Memento

State

Strategy

Example:

User uploads and image.

Based on file type (png, jpg, pdf) then the behavior of the save button changes.

All implementations come from an interface

Template

Defines the algorithm (skeleton) in the Abstract Class

Deferring some steps to the subclasses

Subclasses override steps in the base to change behavior

Example:

Base class has a template method like ProcessPizzaOrder - subclasses changes what goes on that pizza by changing how the steps work like “MakePizza” Method, but wouldn’t change something like “ProcessPayment”

Visitor

Object structure can’t change Operations on the object can

Ex: Currency exchange. The process of calculating a sale would remain the same, but the cost of the item would change based on currency.

ICurrencyVisitor, CurrencyVisitor implements ICurrencyVisitor

Contain methods that can be overloaded based on currency.

Public double Price(Pesos peso) Public double Price(USD dollar)

Price slots into the same process regardless of currency type

Adapter Pattern

Example:

1 Person Interface

1 Male class implements Person

1 Robot class wants to implement the Person interface, but does everything slightly differently

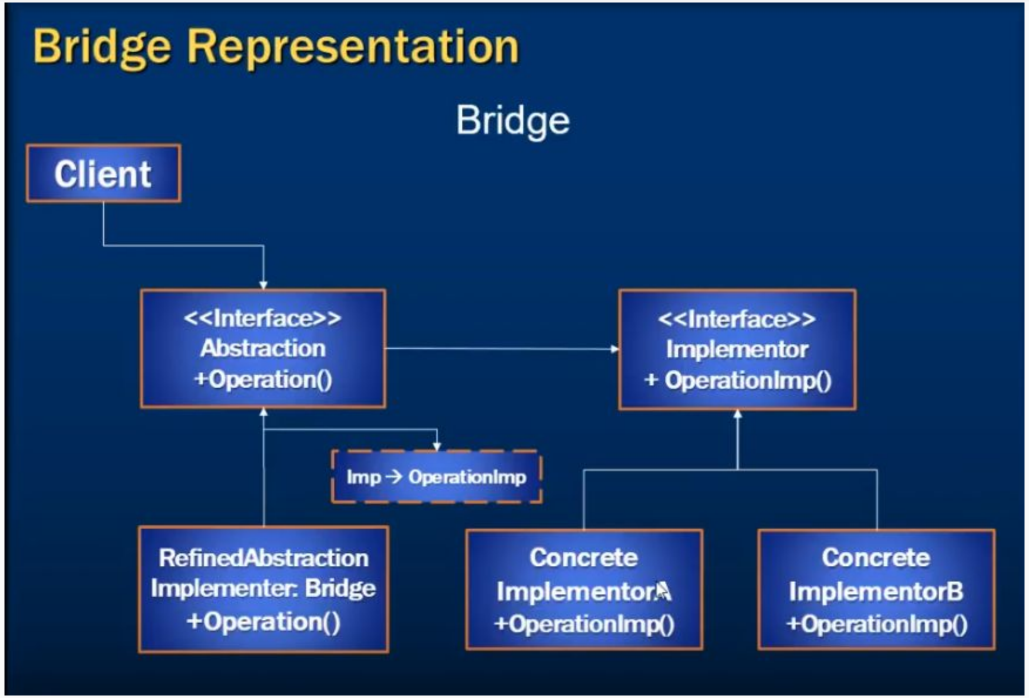
Create an adapter that implements the Person and then calls the Robot methods

Bridge Pattern

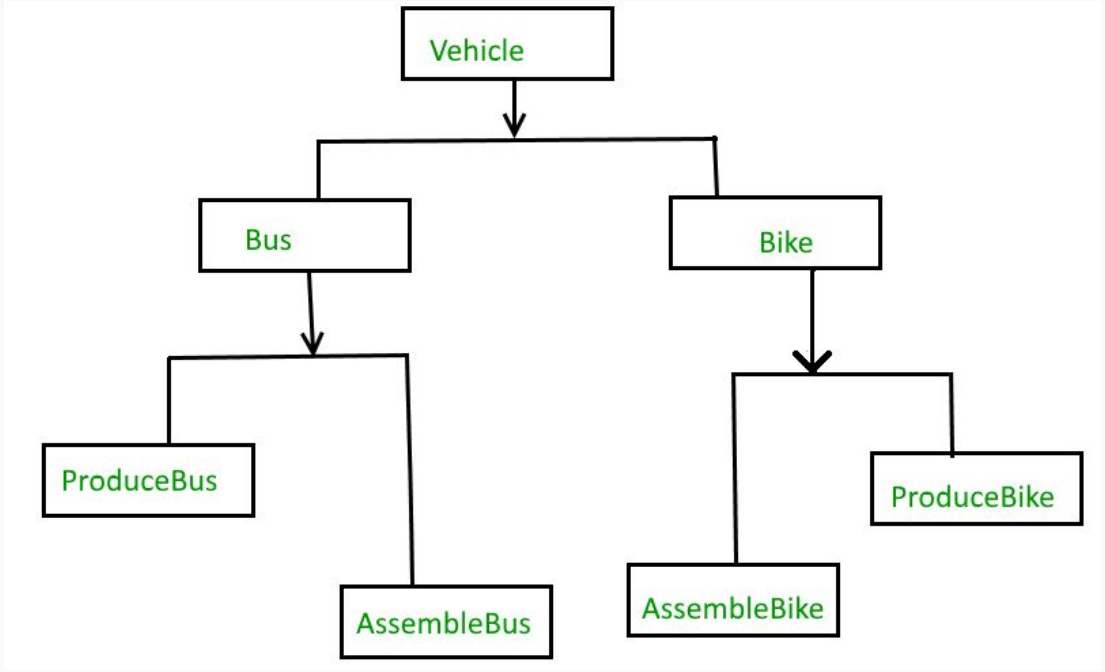
An Interface that “Bridges” between an abstract class and its implementing classes

Decouples the abstract class from the interface, so they can vary independently

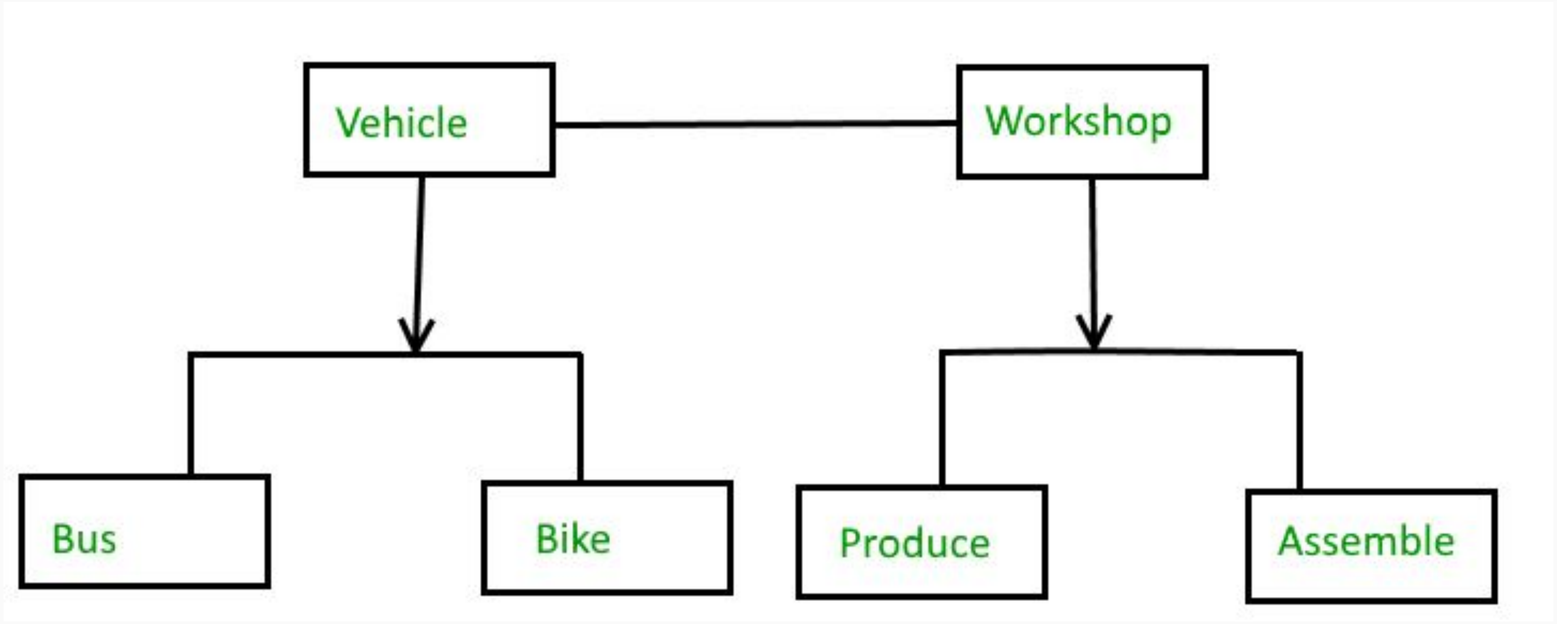
Bridge Pattern



Bridge Pattern



Bridge Pattern



Composite

Decorator Object

Provides alternatives to subclasses for extending functionality

Starbucks - Bad idea - million if statements, a different class for each permutation of drinks, or a class with 2000 properties

Good idea - Each coffee class will have a constructor that takes ICoffee

This allows Espresso:ICoffee object to be sent to Latte:ICoffee

Attributes change (price) based on what gets added to the coffee object

Each ingredient will update the attributes in methods/constructor

Facade

1 Main class (facade) that takes in set parameters like Request + Bank account number

That main class will then use all of the necessary classes/methods based on those parameters

Ex: subclasses could be AccountNumber.cs, Funds.cs, Withdraw.cs

The Facade will then use all of these classes and return information

FlyWeight Pattern

Used to create a large number of similar objects - to reduce time/size of a project by sharing like values such as Intrinsic properties

Intrinsic - shared like Color

If Flyweight(key) exists then return existing flyweight

Else create new flyweight and return it

By keeping track of stored flyweights by a hash or list

Proxy Pattern

A class that limits access to another class

Reasons: Security or object is intensive to create

Example: IATMData is interface

ATMMachine implements IATMData

ATMProxy implements IATMData

ATMProxy for 3rd party software may only allow access to view and not being able to set/retrieve cash

Builder Pattern

Dynamic creation of objects based on interchangeable algorithm

Used to create complex object that has many attributes which can be set in any order OR to deconstruct a complex object

Example: Validations. As you validate a request the builder will add errors/messages to an object for each field the request has

When the process is finished you'll be left with one complex object (like a response)

Factory Pattern

Exposes a method for creating an object, allowing the subclass to control the actual creation

DEMO

Prototype

Use Case:

We have an object that loads data from database. Now we need to modify this data in our program multiple times in multiple places. It’s not a good idea to create the object using the “new” keyword and load all the data again every time.

The better approach would be to clone the existing object into a new one and then do the data manipulation

Singleton

Final point

Know the why and the what to design patterns.

Thanks!

Alex Poling

apoling@manifestcorp.com

614-301-1763

Resources

https://dzone.com/refcardz/design-patterns?chapter=1

https://www.geeksforgeeks.org/bridge-design-pattern/

https://www.youtube.com/watch?v=zRbHlDeon3E

https://www.youtube.com/watch?v=9jIgSslfh_8&list=PLF206E906175C7E07&index=15