Design Patterns

Agenda

- Introduction to Design Patterns
- Types of Design Patterns
- Commonly used
  - Singleton
  - Abstract factory
  - Factory Method
  - Chain of Responsibility

Design Patterns

- Design patterns are a toolkit of **tried and tested solutions** to common problems in software design.
- Design pattern suggests a specific implementation for the specific object-oriented programming problem.
- It describes the problem, the solution, when to apply the solution, and its consequences.

Types of Design pattern

- 23 Gang of Four (GoF) patterns
- All patterns can be categorized by their *intent*, or purpose.
    - **Creational patterns** provide object creation mechanisms that increase flexibility and reuse of existing code.
    - **Structural patterns** explain how to assemble objects and classes into larger structures, while keeping the structures flexible and efficient.
    - **Behavioral patterns** take care of effective communication and the assignment of responsibilities between objects.

Creational Patterns

Abstract Factory: Creates an instance of several families of classes

Builder: Separates object construction from its representation

Factory Method: Creates an instance of several derived classes

Prototype: A fully initialized instance to be copied or cloned

Singleton: A class of which only a single instance can exist

Structural Patterns

Adapter: Match interfaces of different classes

Bridge: Separates an object’s interface from its implementation

Composite: A tree structure of simple and composite objects

Decorator: Add responsibilities to objects dynamically

Facade: A single class that represents an entire subsystem

Flyweight: A fine-grained instance used for efficient sharing

Proxy: An object representing another object

Behavioral Patterns

Chain of Resp.: A way of passing a request between a chain of objects

Command: Encapsulate a command request as an object

Interpreter: A way to include language elements in a program

Iterator: Sequentially access the elements of a collection

Mediator: Defines simplified communication between classes

Memento: Capture and restore an object's internal state

Observer: A way of notifying change to a number of classes

State: Alter an object's behavior when its state changes

Strategy: Encapsulates an algorithm inside a class

Template Method: Defer the exact steps of an algorithm to a subclass

Visitor: Defines a new operation to a class without change

Singleton Design Pattern

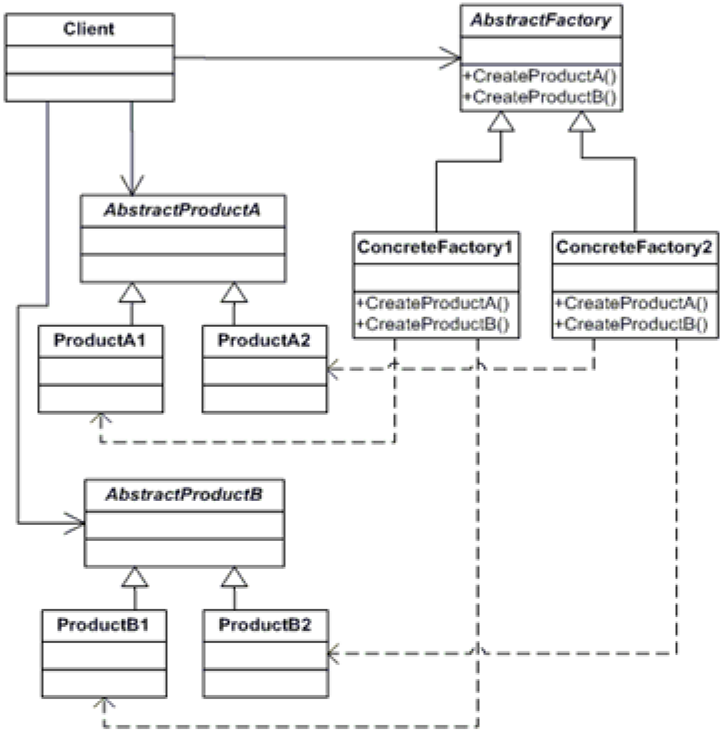
- **Singleton** is a creational design pattern that lets you ensure that a class has only one instance, while providing a global access point to this instance.
- **Problem**:
  - Ensure that a class has just a single instance
  - Provide a global access point to that instance
- **Solution** - All implementations of the Singleton have these two steps in common:
  - Make the default constructor private, to prevent other objects from using the new operator with the Singleton class.
  - Create a static creation method that acts as a constructor. Under the hood, this method calls the private constructor to create an object and saves it in a static field. All following calls to this method return the cached object.

If your code has access to the Singleton class, then it’s able to call the Singleton’s static method. So whenever that method is called, the same object is always returned.

Example: https://www.dofactory.com/net/singleton-design-pattern

Abstract Factory Design Pattern

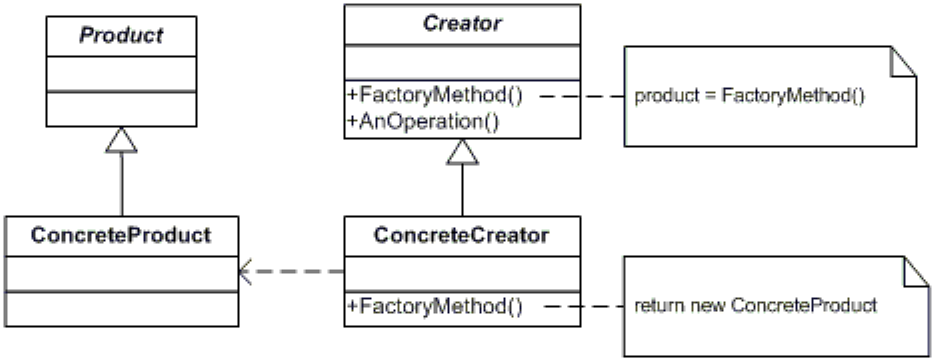
- **Abstract Factory** is a creational design pattern that lets you produce families of related objects without specifying their concrete classes.



Example:
https://www.dofactory.com/net/abstract-factory-design-pattern

Factory Method Design Pattern

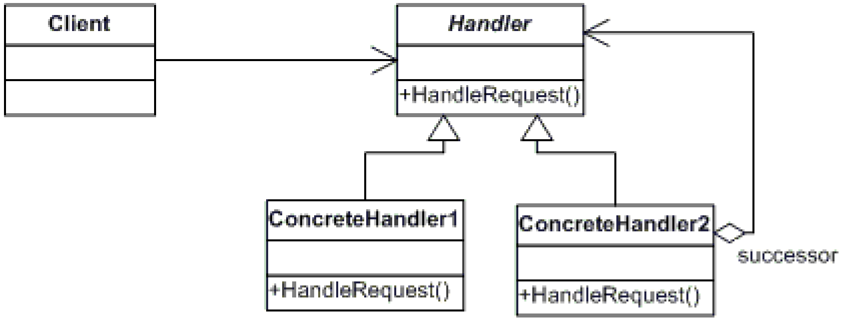
- **Factory Method** is a creational design pattern that provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.
- The Factory Method pattern suggests to replace direct object construction calls with calls to a special *factory* method.



Example: https://www.dofactory.com/net/factory-method-design-pattern

Chain of Resp. Design Pattern

- **Chain of Responsibility** is a behavioral design pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.
- The Factory Method pattern suggests to replace direct object construction calls with calls to a special *factory* method.



Example: https://www.dofactory.com/net/chain-of-responsibility-design-pattern

Q & A