Design Patterns


Design Patterns
Elements of Reusable
Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Fundamental patterns

Creational patterns

Concurrency patterns

Structural patterns

Behavioral patterns

HISTORY OF (DESIGN) PATTERNS

1977 Christopher Alexander introduces the idea of patterns: succesfull solutions to problems

1987 Ward Cunningham and Kent Beck leverage Alexander's idea in the context of an OO language

1987 Eric Gamma's dissertation on importance of patterns and how to capture them

1992 Jim Coplien's book on Advanced C++ Programming Styles and Idioms

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
(gang of four)

Book "Design Patterns:
Elements of Reusable
OO Software"

Patterns Catalogue

Fundamental patterns

Behavioral patterns

Creational patterns

Concurrency patterns

Structural patterns

Patterns Catalogue

Fundamental patterns

Delegation pattern
Interface pattern
Proxy pattern
...

Creational patterns

Abstract factory pattern
Factory method pattern
Lazy initialization pattern
Singleton pattern
...

Structural patterns

Adapter pattern
Bridge pattern
Decorator pattern
...

Behavioral patterns

Chain of responsibility pattern
Iterator pattern
Observer pattern
State pattern
Strategy pattern
Visitor pattern

Concurrency patterns

Active object
Monitor object
Thread pool pattern
...

Patterns Catalogue

- Name
- Intent
- Motivation
- Applicability
- Structure
- Consequences
- Implementation
- Sample Code
- Related Patterns

Factory Method Pattern

Intent

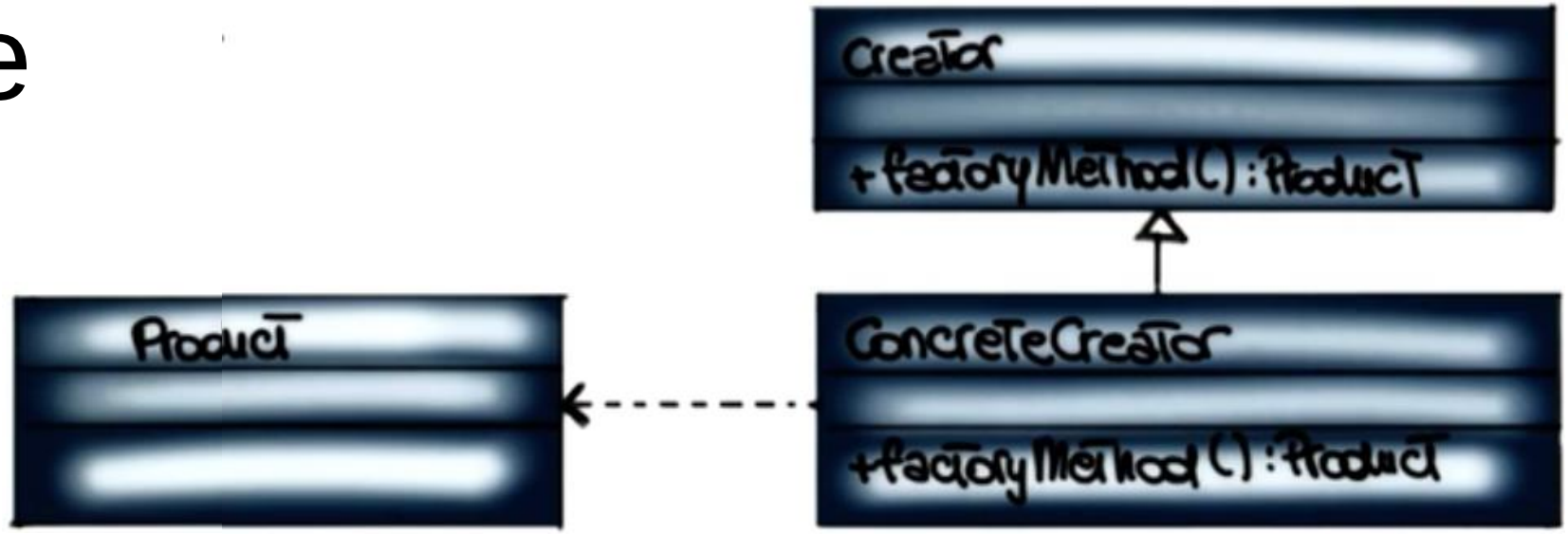
Allows for creating objects without specifying their class, by invoking a factory method (i.e., a method whose main goal is to create class instances)

Applicability

- Class can’t anticipate the type of objects it must create
- Class wants its subclasses to specify the type of objects it creates
- Class needs controlled over the creation of its objects

Factory Method Pattern

Structure



Participants

- Creator: Provides interface for factory method
- ConcreteCreator: Provides method for creating actual object
- Product: Object created by the factory method

Factory Method Pattern

```
public class ImageReaderFactory{
    public static ImageReader createImageReader(InputStream is){
        int imageType = getImageType(is);
        switch(imageType){
            case ImageReaderFactory.GIF
                return new GifReader(is);
            case ImageReaderFactory.JPEG
                return new JpegReader(is);
            // ..........
        }
    }
}
```

Strategy Pattern

Intent

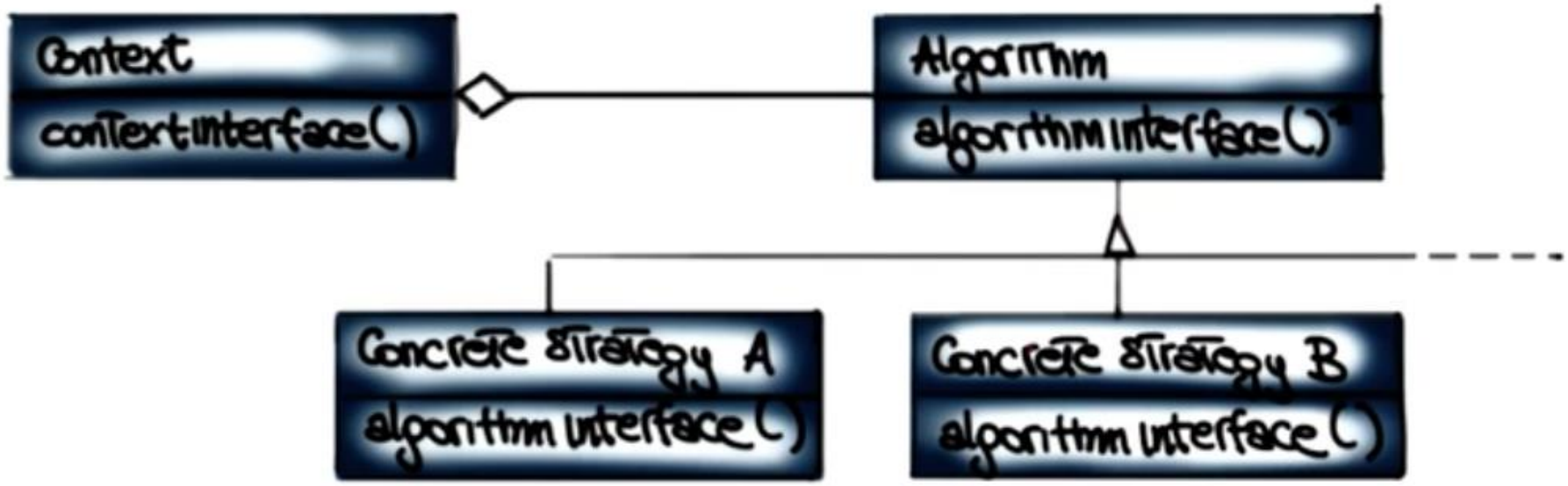
Allows for switching between different algorithms for accomplishing a task

Applicability

- Different variants of an algorithm
- Many related classes differ only in their behavior

Strategy Pattern

Structure



Participants

- Context: interface to outside world
- Algorithm (strategy): common interface for the different algorithms
- Concrete strategy: actual implementation of the algorithm

Strategy Pattern: Example

Program

- Input: text file
- Output: filtered file

Four Filters

- ✓No filtering
- ✓Only words that start with ”t”
- ✓Only words that longer than 5 characters
- ✓Only words that are palindromes

Other Common Patters

Visitor A way of separating an algorithm from an object structure on which it operates

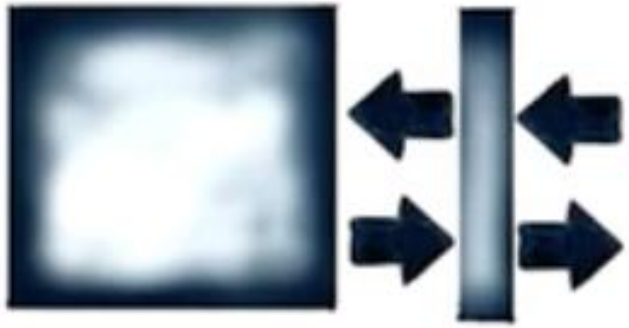
Iterator Access elements of a collection without knowing underlying representation

Decorator A wrapper That adds functionality to a class : stackable

Other Common Patters

Observer

Notify dependents when object changes



Proxy

Surrogate controls access to an object

Choosing a Patters

Approach

- Understand your design context
- Examine the patterns catalogue
- Identify and Study related patterns
- Apply suitable Pattern

Pitfalls

- Selecting wrong patterns
- Abusing Patterns

Imagine that you have to write a class that can have one instance only. Using one of the design patterns that we discussed in this lesson, write the code of a class with only one method (except for possible constructors) that satisfies this requirement. Make sure to call the class Singleton

Imagine that you have to write a class that can have one instance only. Using one of the design patterns that we discussed in this lesson, write the code of a class with only one method (except for possible constructors) that satisfies this requirement. Make sure to call the class Singleton

```
public class Singleton {
    private static Singleton instance;
    private Singleton() {}
    public static Singleton factory() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```

NEGATIVE DESIGN PATTERNS

Also in Christopher Alexander's book

How not to (design, manage, etc.)

Also called anti-patterns and bad smells