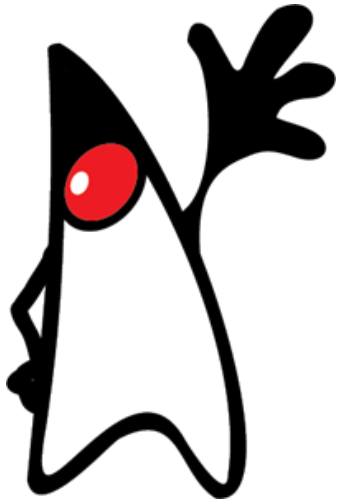
Design patterns



What are "patterns"?

- For a software design problem, there is not always one right solution, but sometimes several possibilities.
- For common problems, 'patterns' offer a good solution; they are **best practices**
- Patterns are a **means of communication**: between designers and programmers: easy to know what you are talking about.

Patterns - advantages

- **Better Maintainability/Faster Development Time**: Standard pattern known immediately
- **Reusability**: an entire solution is reused
- **Language-independent**: widely applicable, independent of any programming language

Patterns: properties

- **Name**: means of communication
- **Problem**: description of the problem it solves
- **Solution**: description of how the problem is solved
- **Consequences**: consequences of applying the pattern

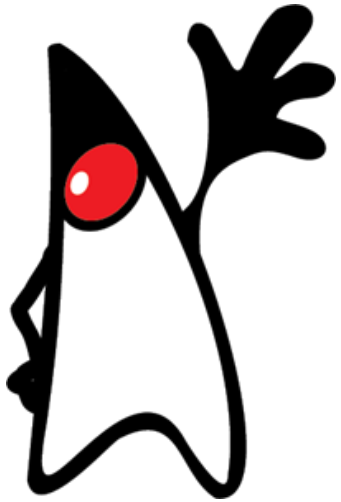
What are patterns?

- We already saw in UML:
  - **GRASPatterns**
    - General Responsibility Assignment Software Patterns
- In this lesson:
  - GOF **Design** Patterns
    - GOF
      - = Gang Of Four
      - = the authors of the book Design Patterns: Elements of Reusable Object-Oriented Software
      - = Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

Contents

- Designterns_theory
  - GoF - design patterns explained
- Assignment design patterns 2021-2022

Design Patterns Theory



Contents

- Types of GoF Design patterns
- Worked out patterns:
  - Singleton
  - Observer
  - Strategy
  - Decorator

3 types of GOF Design patterns

- In the GOF design patterns we distinguish 3 types:
  - **Creational** patterns
    - Mechanisms for creating class objects
  - **Structural** patterns
    - About relationships between entities and how they work together
  - **Behavioral** patterns
    - For communication between entities

GOF Design patterns

- In the figure below you find an overview of all GOF design patterns. In this lesson we will concentrate on 4 of these patterns:
  - Singleton
  - Observer
  - Strategy
  - Decorator

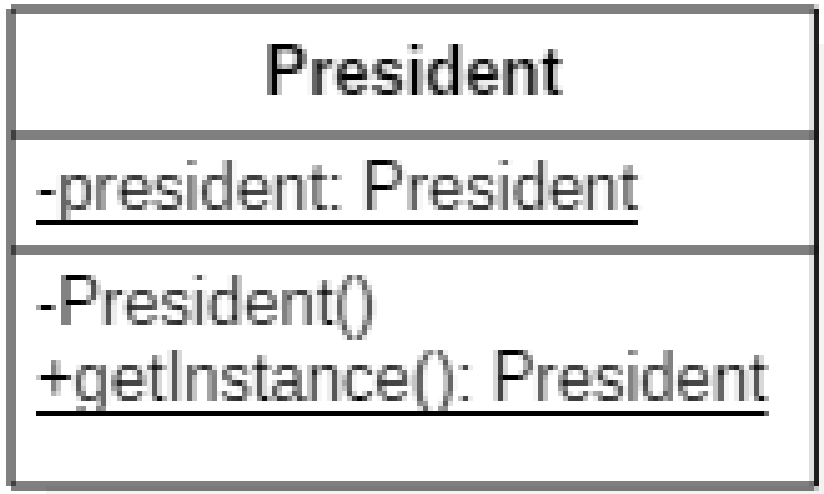
By Purpose				
		Creational	Structural	Behavioral
By Scope	Class	• Factory Method	• Adapter (class)	• Interpreter • Template Method
	Object	• Abstract Factory • Builder • Prototype ★ Singleton	• Adapter (object) • Bridge • Composite ★ Decorator • Façade • Flyweight • Proxy	• Chain of Responsibility • Command • Iterator • Mediator • Memento ★ Observer • State ★ Strategy • Visitor

Singleton (object - creational)

Movie

- Singleton Pattern:
  - **Issue**: How can you ensure that only one instance of a class is created?
  - **Solution**: Give the class a private instance of itself and a private constructor. You can only access the private instance through a public method. This method creates the instance if it is null, and then passes this single instance.
  - **Note**: only use this pattern if you are 100% sure that only one object of your (singleton) class is needed at any time.

Singleton - code example



Singleton - code example


```
public final class President {

    private static President

    private President() {
      System.out.println("A singleton object is created.");
    }

    public static President getInstance() {
        if (president == null) {
            president = new President();
        }
        return president;
    }
}
```

Singleton - code example


```
President first, second;
first = President.getInstance();
second = President.getInstance();

If (first==second){
      System.out.println("The two singleton variables refer to the same
object.");
}
else {
      System.out.println("This is not possible in principle.");
}
```

Design patterns - other

- Singleton is the simplest design pattern, but not the only one...

Me: So! Have you ever heard of a book called Design Patterns?
Them: Oh, yeah, um, we had to, uh, study that back in my software engineering class. I use them all the time.
Me: Can you name any of the patterns they covered?
Them: I loved the Singleton pattern!
Me: OK. Were there any others?
Them: Uh, I think there was one called the Visitater.
Me: Oooh, that's right! The one that visits potatoes. I use it all the time. Next!!!

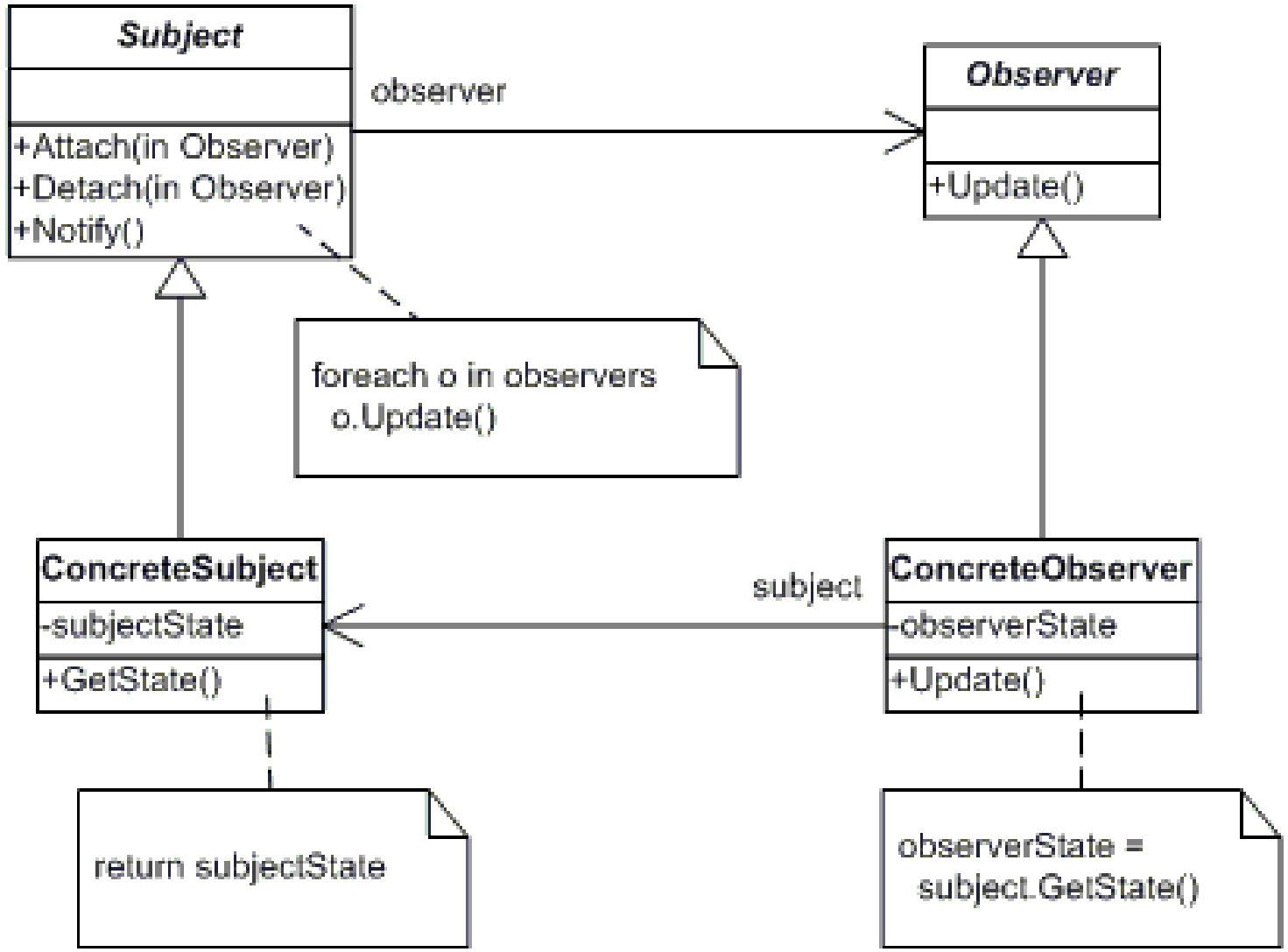
I actually use this as a question now. If they claim expertise at Design Patterns, and they can ONLY name the Singleton pattern, then they will ONLY work at some other company.

Observer (behavioural)

Video **(up to minute 14, you may skip the part that follows about threats)**

- Observer Pattern:
  - **Customer's dilemma**: How can multiple objects be kept informed about the state of one particular object?
  - **Solution**: We call the object in which the other objects are interested "subject". This "subject" keeps a list of all interested objects, which we call "observers". The "subject" notifies the "observers" automatically about every state change with a method call (in our example update())

Observer pattern - general



Class diagram observer pattern

Observer pattern - general

Details observer pattern:

- **Subject**: abstract class or interface to maintain list of observers, can add and remove observers
  - ***Attach*** *- add the observer (in the parameter) to the ArrayList of observers*
  - ***Detach*** *- remove observer (in the parameter) from the observers ArrayList*
  - ***Notify*** *- notify all observers in the ArrayList of a change by looping through all observers in the ArrayList and* calling *the update function of each object in the ArrayList*
- **ConcreteSubject** inherits/implements subject and contains the state that the observers are interested in. Sends notification to all observers by calling the notify function (see Subject class) when the state changes (setState).
  - ***getState*** *- Returns state of subject*

Observer pattern - general

Details observer pattern (continued):

- **Observer** abstract class or interface, defines an updating method for all observers so that they can receive an update notification from subject.
  - ***update*** *- abstract function, will be overridden by concrete observers*
- **ConcreteObserver** keeps a reference to subject in order to be able to retrieve the state of subject when he receives a notification
  - ***update*** *- (overriding) When the subject calls this function, the concreteObserver calls the getState operation of the subject in order to obtain information about its state.*

Observer pattern - example code

- You can check the example code from the video here:

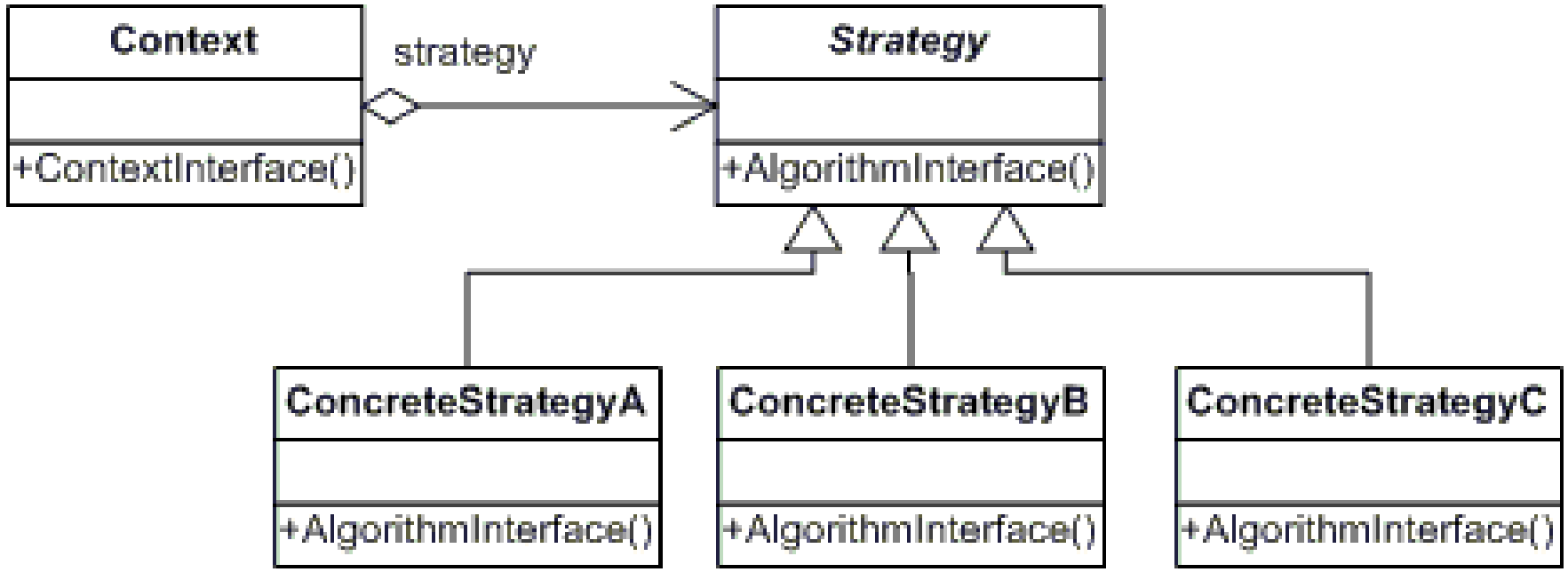
http://www.newthinktank.com/2012/08/observer-design-pattern-tutorial/

Strategy (behavioural)

Movie

- Strategy Pattern:
  - **Customer's dilemma**: How can you ensure that at runtime it can be decided which algorithm should be used to determine the output of a particular method?
  - **Solution**: Define, using an interface, a family of algorithms that each give a different implementation to one particular method.

Strategy pattern - general



Class diagram strategy pattern

Strategy pattern - general

- **Strategy** abstract class or interface for the algorithm/strategy of which all concrete algorithms inherit. It provides a common interface for all the concrete algorithms/strategies. All the abstract functions of the Strategy class must be overridden by the ConcreteStrategy classes.
- **ConcreteStrategy** In this class we implement the algorithm.
- **Context** can be configured with one or more ConcreteStrategies. It uses this ConcreteStrategy via the interface defined in Strategy.

Strategy pattern - example code

- You can check the example code from the video here:

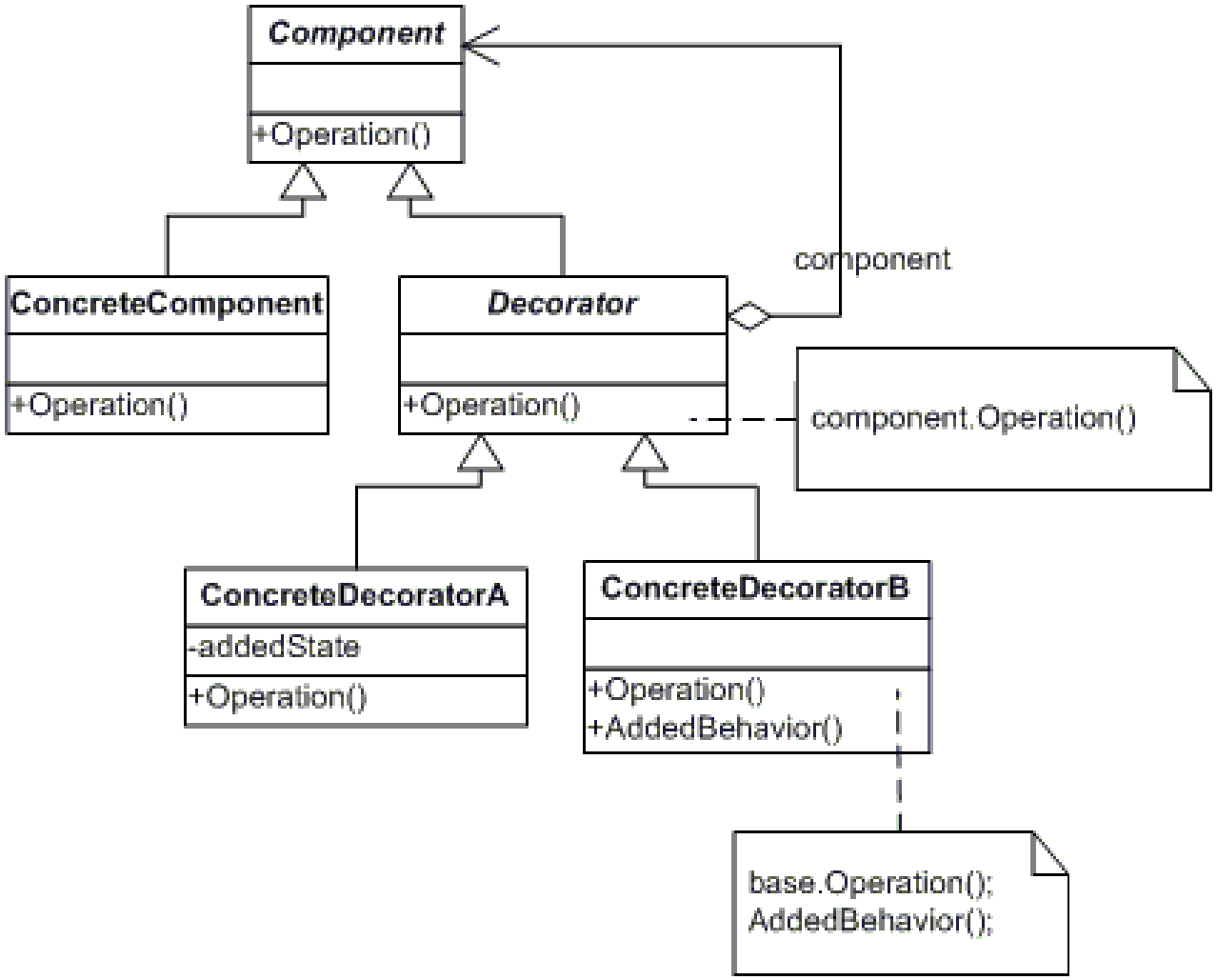
http://www.newthinktank.com/2012/08/strategy-design-pattern-tutorial/

Decorator (strutural)

Movie

- Decorator Pattern:
  - Problem definition: You want the functionality of inheritance but you want to be able to change this @runtime. You want to dynamically assign responsibilities (methods) to an object.

Decorator pattern - general



Class diagram decorator pattern

Decorator pattern - general

- **Component** abstract class or interface that defines an interface for components.
- **ConcreteComponent** concrete implementation of the component class. Defines an object to which **additional** responsibilities / properties can be assigned.
- **Decorator** inherits attributes and operations from a Component and maintains a reference to an object of the class
- **ConcreteDecorator** This is the class that we use to assign additional responsibilities, additional properties and/or other versions of existing responsibilities to a component.

Decorator pattern - example code

- You can check the example code from the video here:

http://www.newthinktank.com/2012/09/decorator-design-pattern-tutorial/

Assignments on Canvas

- Assignment design patterns 2021-2022
  - Goal = get design patterns working in a restaurant application
  - In the start folder you will also find in Canvas, the tests that should all pass if you have implemented the design patterns correctly
- Assignment web application
  - Objective = to create a web application with the technology you have learned in the lessons
  - Start = the project in which you got the design patterns working
- Tips
  - Make regular backups and new versions as soon as you have updated a functionality, so that you can always fall back on a working project.
- GOOD LUCK