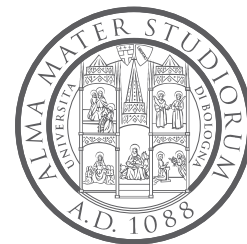


# Design Patterns part 2

Davide Rossi

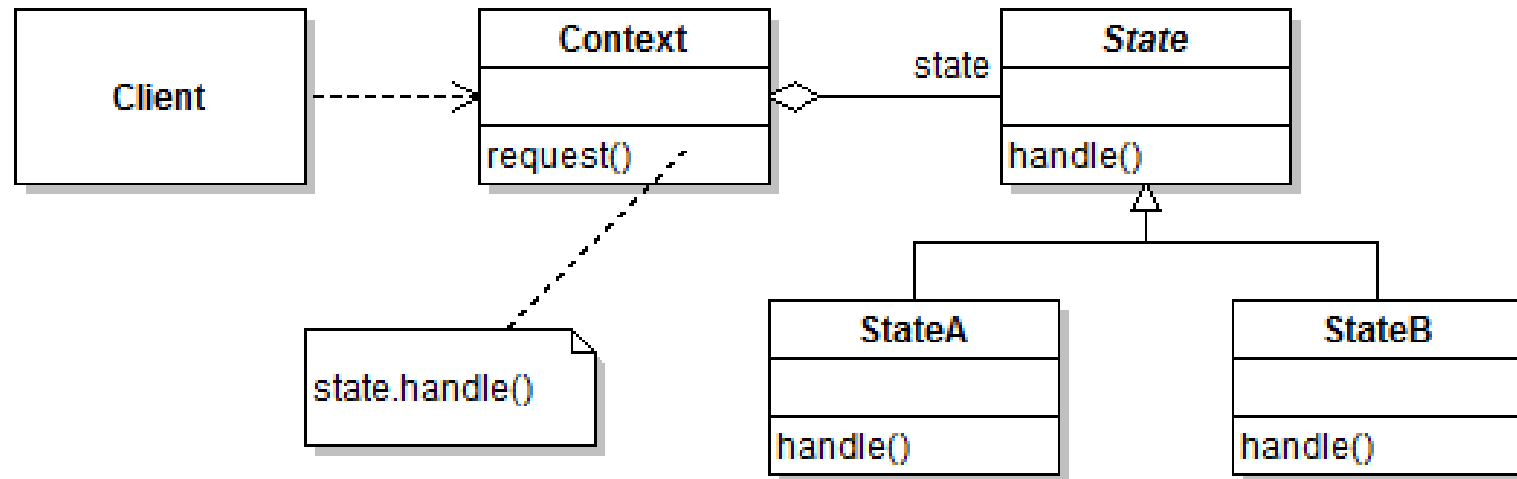
Dipartimento di Informatica – Scienze e Ingegneria  
Università di Bologna



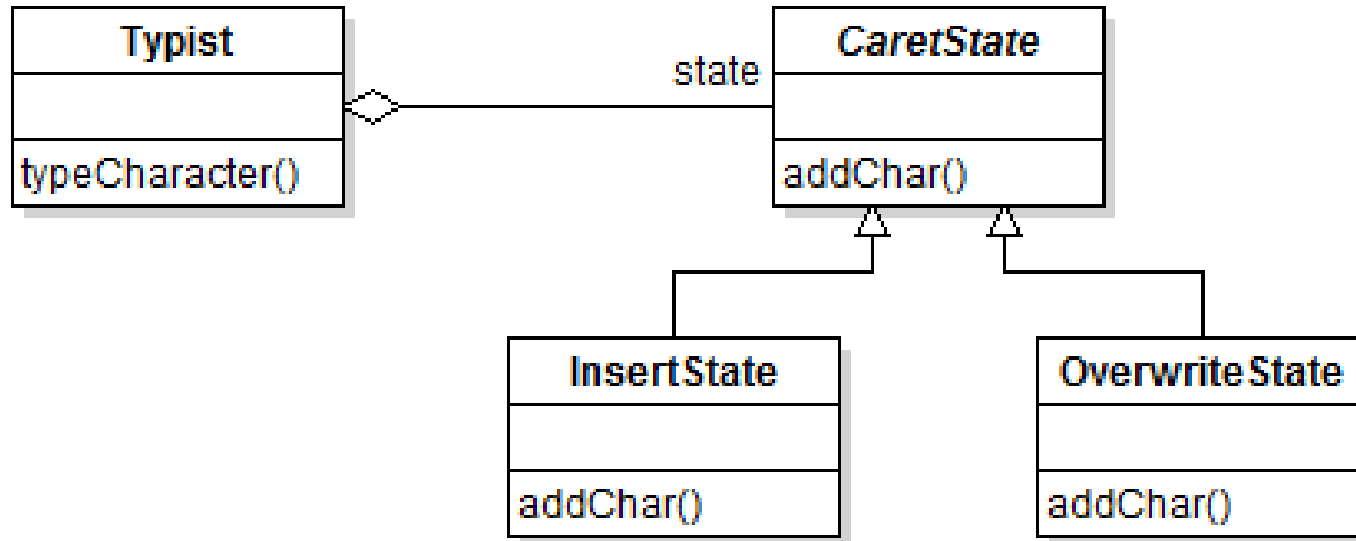
# GoF: State

- Problem: how can I change the behavior of an object depending on its state with a high-quality solution?
- State: allow an object to alter its behavior when its internal state changes.
- The object will appear to change its class.

# GoF: State



# State example

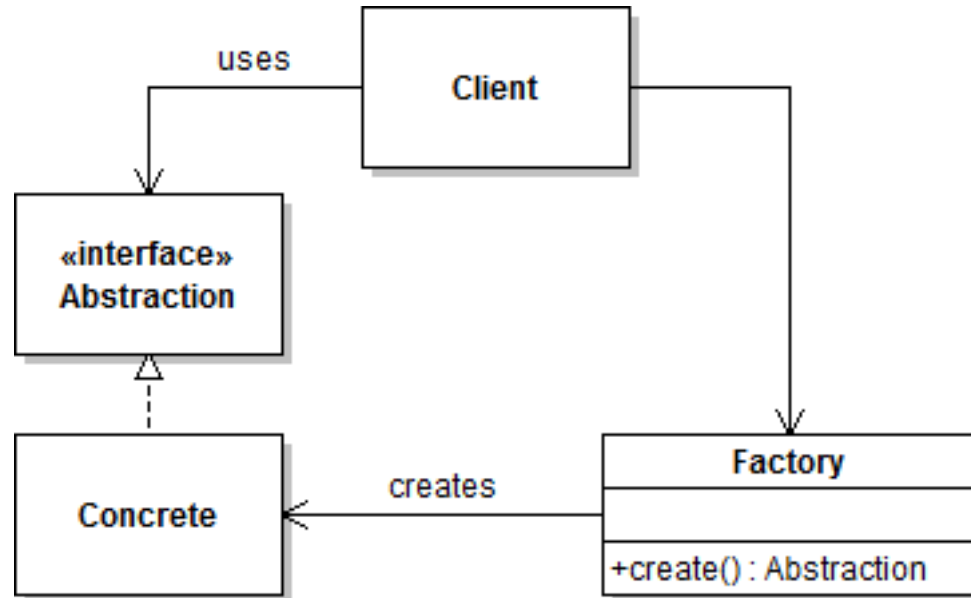


# New considered armful

- Allocates memory each time a client needs a reference
- No control on instances lifecycle
- Changes in the constructor of the concrete class breaks clients (**violates OCP**)
- Creates a dependency between the user and the concrete class implemented by the reference (**violates DIP**)

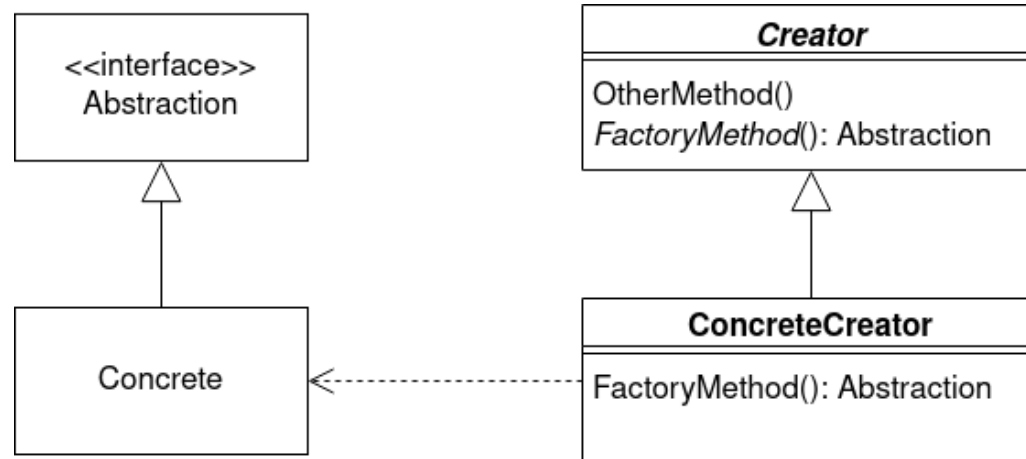
# Factory

- Decouples client from instantiation process
- Refers to the newly created object through a common interface



# GoF: Factory method

- Define an interface for creating an object, but let subclasses decide which class to instantiate
- Factory Method lets a class defer instantiation to subclasses

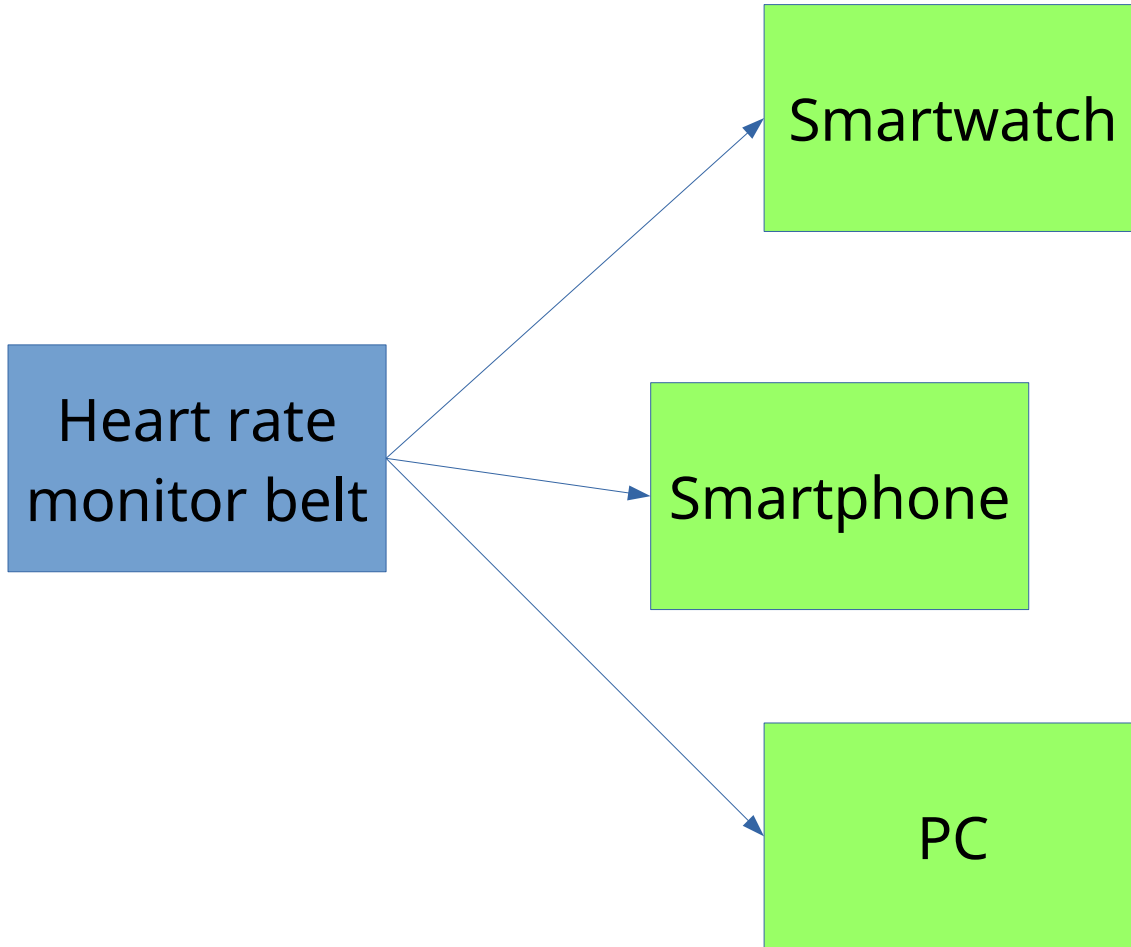


# Factory Method

- A popular variation collapses the abstraction and the creator putting the factoryMethod inside the abstraction (that becomes an abstract class)
- Java API examples:
  - `java.util.Calendar#getInstance()`
  - `java.text.NumberFormat#getInstance()`
  - `java.nio.charset.Charset#forName()`



# The notification problem

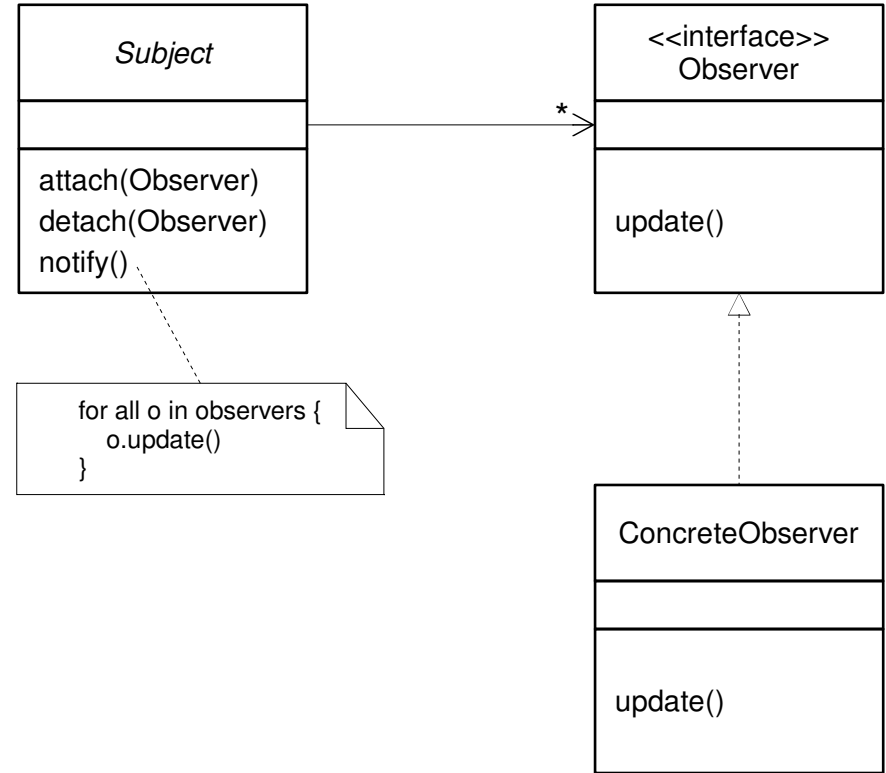


# The notification problem

- New observers can appear at a later time
- New types of observers can appear at a later time
- When a class “notifies” another it is exposed to its interface (thus it depends on that interface)
- ISP: The dependency of one class to another one should depend on the smallest possible interface
- DIP: Depend upon Abstractions
- PV: Identify points of predicted variation or instability; assign responsibilities to create a stable interface around them.

# GoF: Observer

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.



# Observer in Java

```
java.util.Observer
```

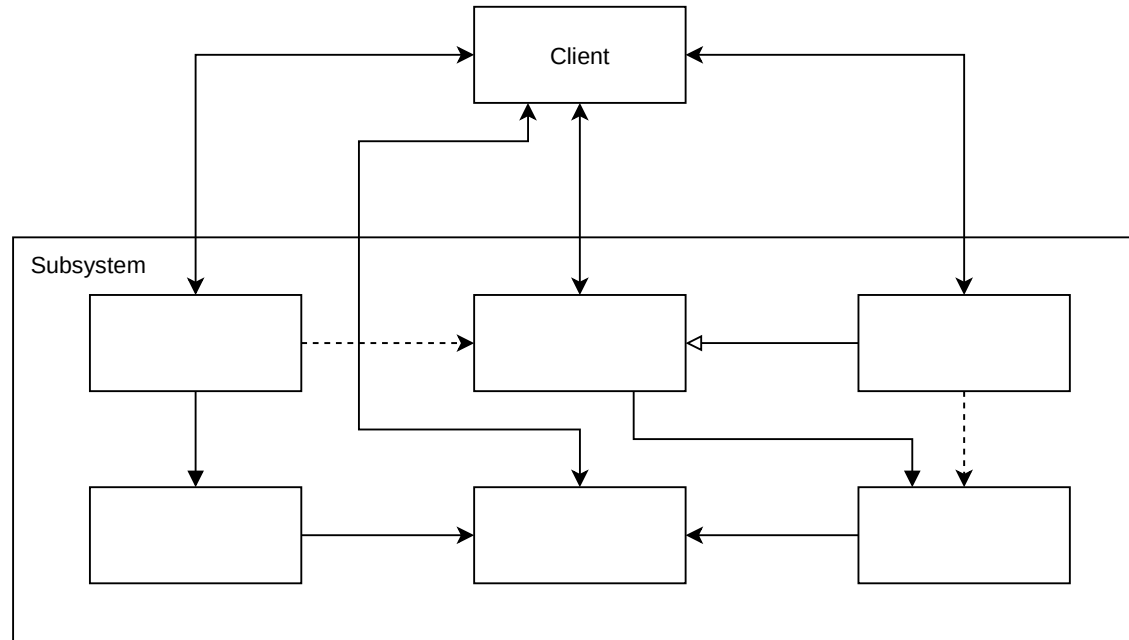
```
public interface Observer {  
    void update(Observable o, Object arg)  
}
```

```
java.util.Observable
```

```
public class Observable {  
    public void addObserver(Observer o);  
    public void deleteObserver(Observer o);  
    public void notifyObservers();  
    protected void setChanged();  
    ...  
}
```

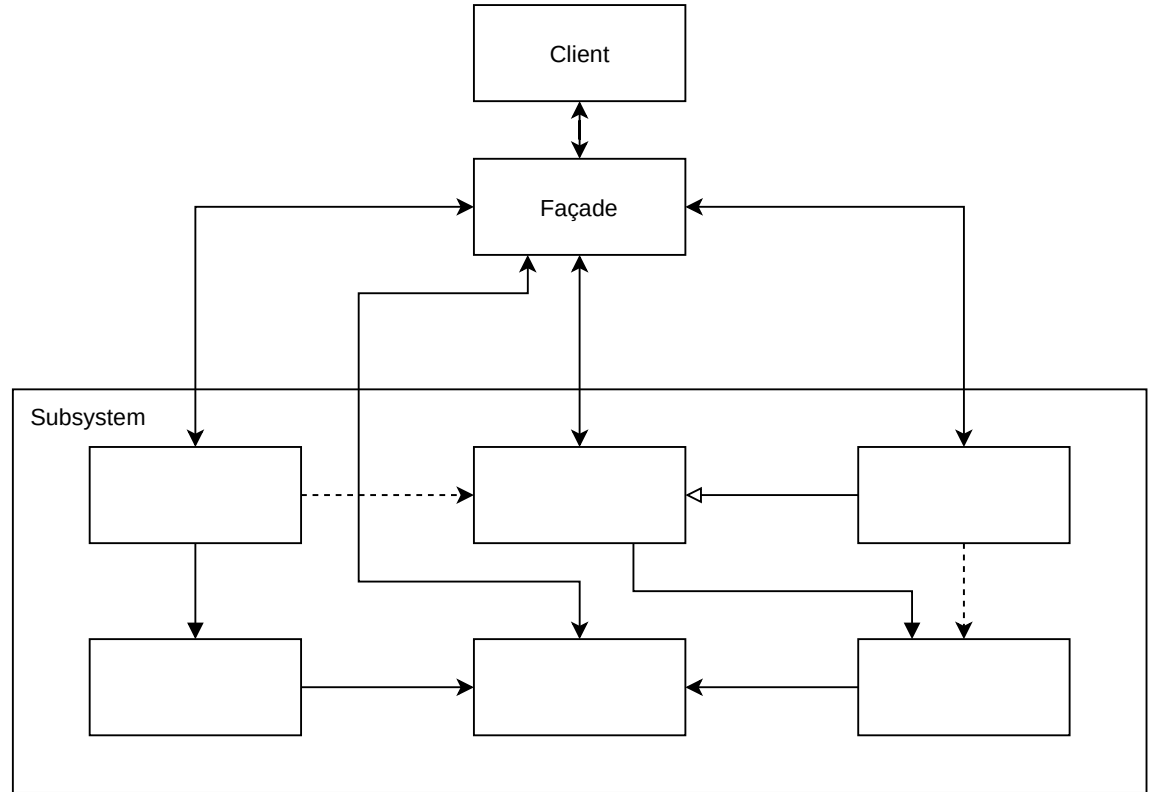
# GoF: Façade

- Problem: how can I isolate a client from the internal complexity of a subsystem?



# GoF: Façade

- Façade: provide a unified interface to a set of interfaces in a subsystem.
- Facade defines a higher-level interface that makes the subsystem easier to use.



# Resources

## Books

- Eric Freeman & Elisabeth Robson, *Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software* (2nd Edition), O'Reilly

## Online:

- <http://www.vincehuston.org/dp/>
- <http://www.oodesign.com/>
- <https://refactoring.guru/design-patterns/>
- <http://www.informit.com/articles/article.aspx?p=1404056>