Dr. Michael Eichberg
Software Engineering
Department of Computer Science
Technische Universität Darmstadt

Software Engineering

# The Observer Design Pattern

For details see Gamma et al. in “Design Patterns”

TECHNISCHE
UNIVERSITÄT
DARMSTADT

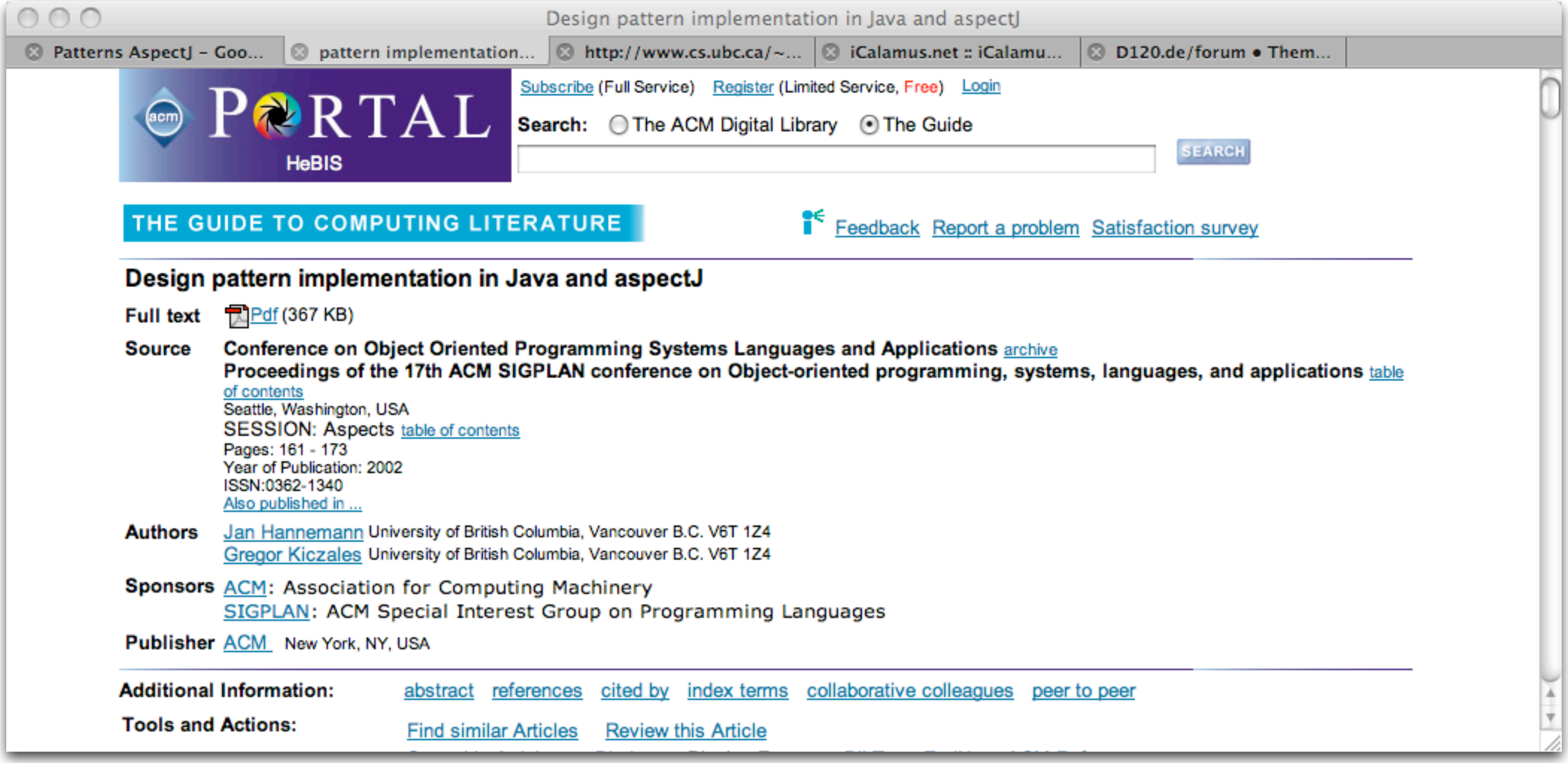
# Observer Design Pattern

## Intent

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

# The Observer Design Pattern

## Alternative Implementation using AspectJ


Design pattern implementation in Java and aspectJ
Patterns AspectJ - Goo...
pattern implementation...
http://www.cs.ubc.ca/~...
iCalamus.net :: iCalamu...
D120.de/forum • Them...
acm
PORTAL
HeBIS
SEARCH
THE GUIDE TO COMPUTING LITERATURE

Subscribe (Full Service) Register (Limited Service, Free) Login

Search: ○ The ACM Digital Library ● The Guide

Feedback Report a problem Satisfaction survey

**Design pattern implementation in Java and aspectJ**

| | |
|---|---|
| **Full text** | Pdf (367 KB) |
| **Source** | **Conference on Object Oriented Programming Systems Languages and Applications** archive<br>**Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications** table of contents<br>Seattle, Washington, USA<br>SESSION: Aspects table of contents<br>Pages: 161 - 173<br>Year of Publication: 2002<br>ISSN:0362-1340<br>Also published in ... |
| **Authors** | Jan Hannemann University of British Columbia, Vancouver B.C. V6T 1Z4<br>Gregor Kiczales University of British Columbia, Vancouver B.C. V6T 1Z4 |
| **Sponsors** | ACM: Association for Computing Machinery<br>SIGPLAN: ACM Special Interest Group on Programming Languages |
| **Publisher** | ACM New York, NY, USA |

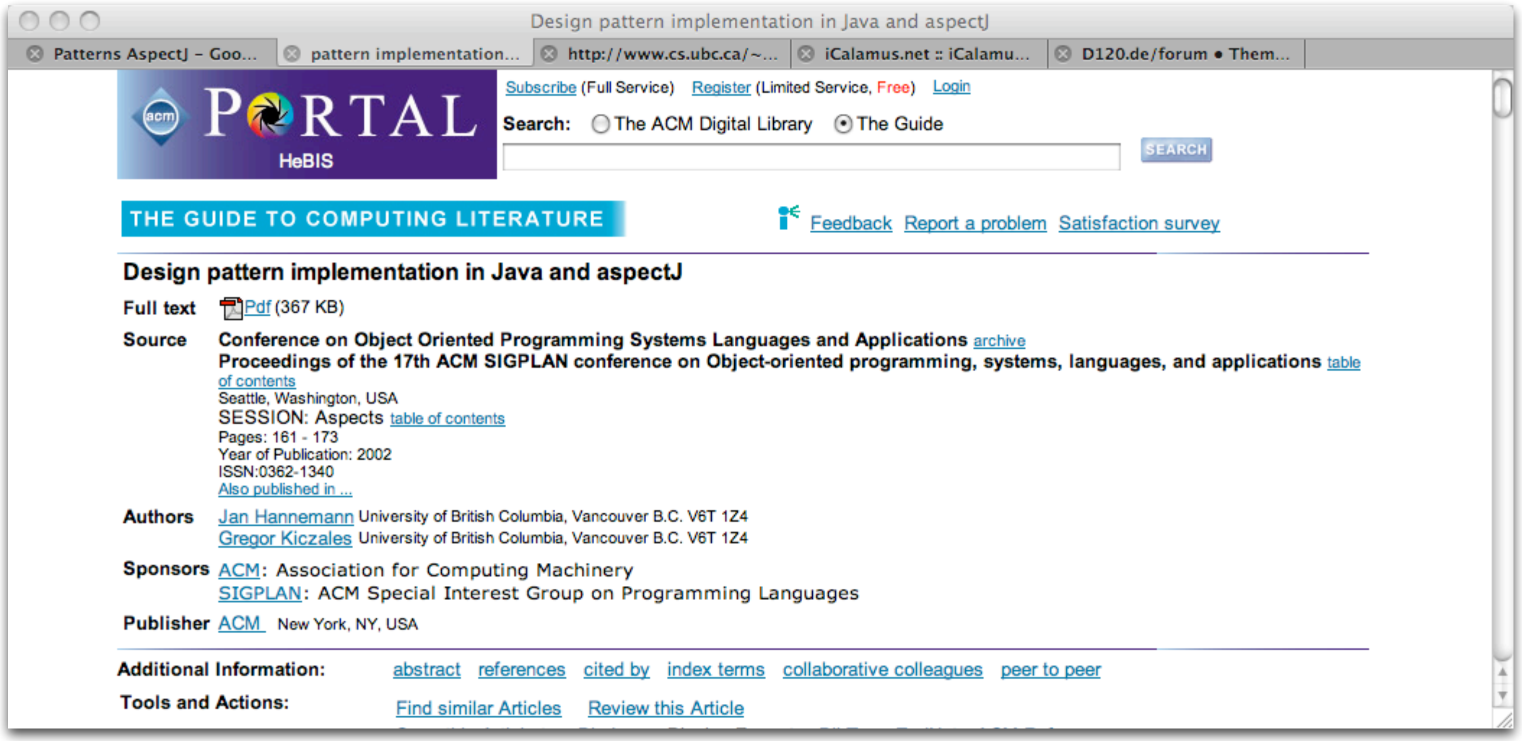
**Additional Information:** abstract references cited by index terms collaborative colleagues peer to peer

**Tools and Actions:** Find similar Articles Review this Article

# The Observer Design Pattern

## Alternative Implementation using AspectJ

- We want to...
  - **avoid the decision between Push or Pull mode observers**
  - better **support observers interested only in specific events**


Design pattern implementation in Java and aspectJ
Patterns AspectJ – Goo...
pattern implementation...
http://www.cs.ubc.ca/~...
iCalamus.net :: iCalamu...
D120.de/forum • Them...
acm
PORTAL
HeBIS
Subscribe (Full Service) Register (Limited Service, Free) Login
Search: ○ The ACM Digital Library ● The Guide
SEARCH
THE GUIDE TO COMPUTING LITERATURE
Feedback Report a problem Satisfaction survey

**Design pattern implementation in Java and aspectJ**

**Full text** Pdf (367 KB)

**Source** **Conference on Object Oriented Programming Systems Languages and Applications** archive
**Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications** table of contents
Seattle, Washington, USA
SESSION: Aspects table of contents
Pages: 161 - 173
Year of Publication: 2002
ISSN:0362-1340
Also published in ...

**Authors** Jan Hannemann University of British Columbia, Vancouver B.C. V6T 1Z4
Gregor Kiczales University of British Columbia, Vancouver B.C. V6T 1Z4

**Sponsors** ACM: Association for Computing Machinery
SIGPLAN: ACM Special Interest Group on Programming Languages

**Publisher** ACM New York, NY, USA

**Additional Information:** abstract references cited by index terms collaborative colleagues peer to peer

**Tools and Actions:** Find similar Articles Review this Article

# The Observer Design Pattern

## Alternative Implementation using AspectJ

### *Parts Common to Potential Instantiations of the Pattern*

Will be implemented in a reusable ObserverProtocol aspect.

1. The existence of **Subject** and **Observer** *roles*
   (i.e. the fact that some classes act as Observers and some as Subjects)
2. Maintenance of a mapping from **Subject**s to **Observer**s
3. The general update logic: **Subject** changes trigger Observer updates

### *Parts Specific to Each Instantiation of the Pattern*

4. Which classes can be **Subject**s and which can be **Observer**s
5. A set of changes of interest on the **Subject**s that trigger updates on the **Observer**s
6. The specific means of updating each kind of **Observer** when the update logic requires it

# The Observer Design Pattern

## Alternative Implementation using AspectJ

```
public abstract aspect ObserverProtocol {

   // Realization of the Roles of the Observer Design Pattern
   protected interface Subject  { }
   protected interface Observer { }

  ...
}
```

The part common to instantiations of the pattern.

# The Observer Design Pattern

## Alternative Implementation using AspectJ

```
public abstract aspect ObserverProtocol {
    ...
    // Mapping and Managing Subjects and Observers
    private WeakHashMap<Subject, List<Observer>> perSubjectObservers;
    protected List<Observer> getObservers(Subject s) {
        if (perSubjectObservers == null)
            perSubjectObservers = new WeakHashMap<Subject, List<Observer>>()
        List<Observer> observers = perSubjectObservers.get(s);
        if ( observers == null ) {
            observers = new LinkedList<Observer>();
            perSubjectObservers.put(s, observers);
        }
        return observers;
    }
    public void addObserver(Subject s,Observer o){
        getObservers(s).add(o);
    }
    public void removeObserver(Subject s,Observer o){
        getObservers(s).remove(o);
    }
  ...
```

The part common to instantiations of the pattern.

# The Observer Design Pattern

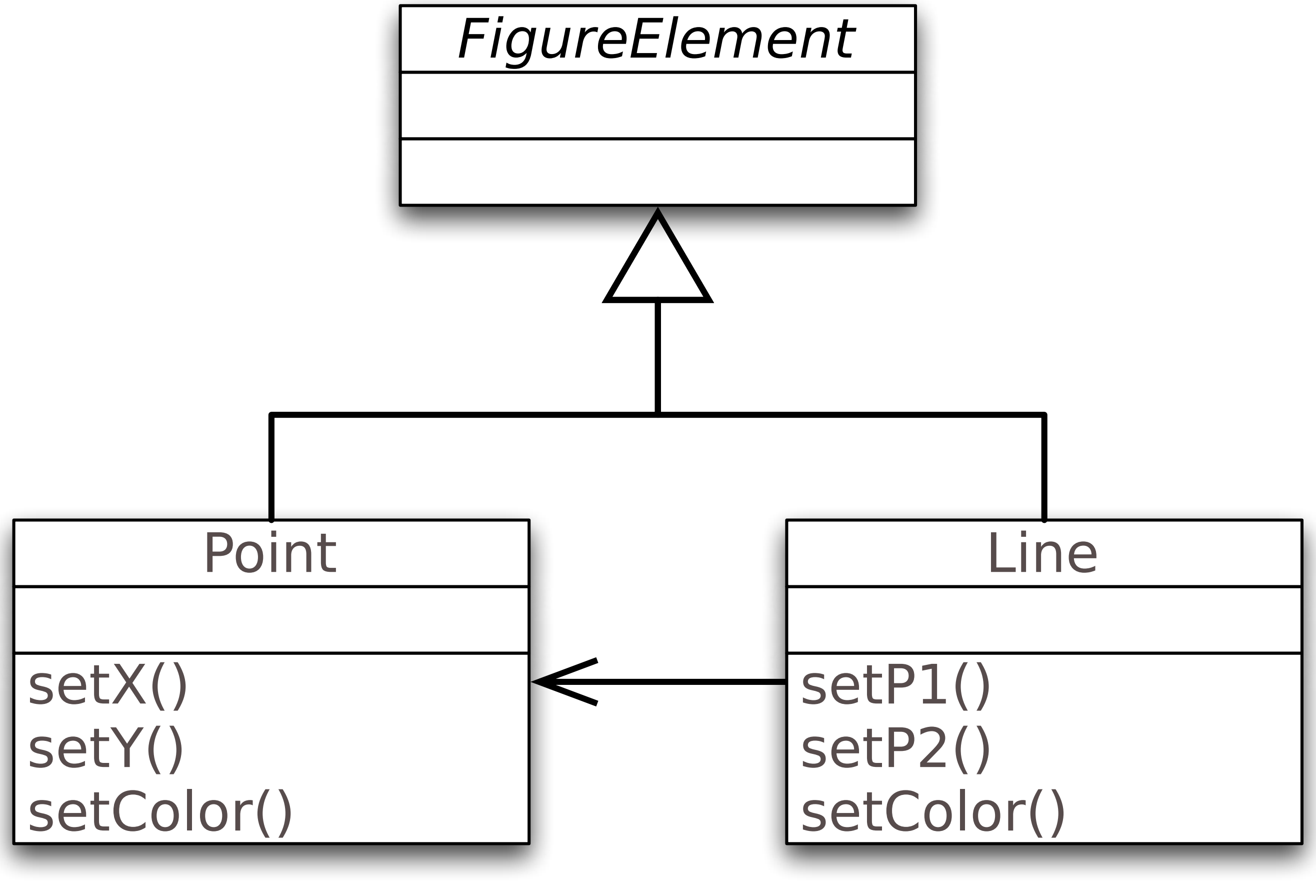
## Alternative Implementation using AspectJ

```
public abstract aspect ObserverProtocol {
    ...
    // Notification related functionality
    abstract protected pointcut subjectChange(Subject s);

    abstract protected void updateObserver(Subject s, Observer o);

    after(Subject s): subjectChange(s) {
        Iterator<Observer> iter = getObservers(s).iterator();
        while ( iter.hasNext() ) {
            updateObserver(s, iter.next());
        }
    }
}
```

The part common to instantiations of the pattern.

# The Observer Design Pattern

## Alternative Implementation using AspectJ - Example



# The Observer Design Pattern

## Alternative Implementation using AspectJ - Example

### Task: Observe Changes of the Color

```
public aspect ColorObserver extends ObserverProtocol {

    declare parents: Point  implements Subject;
    declare parents: Line   implements Subject;
    declare parents: Screen implements Observer;

    protected pointcut subjectChange(Subject s):
        (call(void Point.setColor(Color)) ||
         call(void Line.setColor(Color)) ) && target(s);

    protected void updateObserver(Subject s, Observer o) {
        ((Screen)o).display("Color change.");
    }
}
```

### To create a mapping between an Observer and a Subject:

```
ColorObserver.aspectOf().addObserver(P, S);
```

# The Observer Design Pattern

## Alternative Implementation using AspectJ - Assessment

- **Locality**
  All code that implements the Observer pattern is in the abstract and concrete observer aspects, none of it is in the participant classes; there is no coupling between the participants. Potential changes to each Observer pattern instance are confined to one place.
- **Reusability**
  The core pattern code is abstracted and reusable. The implementation of ObserverProtocol is generalizing the overall pattern behavior. The abstract aspect can be reused and shared across multiple Observer pattern instances.

# The Observer Design Pattern

## Alternative Implementation using AspectJ - Assessment

- **Composition transparency**
  Because a pattern participant's implementation is not coupled to the pattern, if a Subject or Observer takes part in multiple observing relationships their code does not become more complicated and the pattern instances are not confused. Each instance of the pattern can be reasoned about independently.
- **(Un)pluggability**
  It is possible to switch between using a pattern and not using it in the system.

# Observer Design Pattern

How it is implemented depends on the available programming language mechanisms; the consequences may also change!

Programming Languages ⟺ Design Pattern