What is a Design Pattern

- … describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever using it the same way twice [Christopher Alexander quoted by Gamma, et. al.]

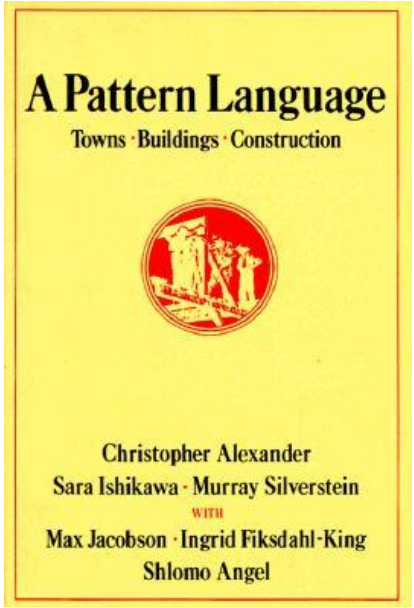
- A pattern is a problem-solution pair, which can be used in different contexts, together with an advise, on how it has to be used in a new situation

What is a Design Pattern

- **Reusable solutions to recurring problems that we encounter during software development [Mark Grand]**
- **Documentation of a solution for a given (analysis, design or implementation) problem**
- **Description of a frequently occurring structure of cooperating components, which solves a general design problem in a particular context**

Design Patterns: History

- **Christopher Alexander**
  - Published a pattern book 1970
  - Pattern language in the realm of urban planning and building architecture
  - 253 patterns for buildings and public spaces
  - Each pattern provided a general solution to a reoccurring design problem in a particular context



Design Patterns: History

- **Low Sill Pattern**
  - Context
    - Windows are being planned for a wall.
  - Problem
    - How high should the windowsill be from the floor? A windowsill that is too high cuts you off from the outside world. One that is too low could be mistaken for a door and is potentially unsafe.
  - Solution
    - Design the windowsill to be 12 to 14 inches from the floor. On upper floors, for safety, make them higher, around 20 inches. The primary function of a window is to connect building occupants to the outside world. The link is more meaningful when both the ground and horizon are visible when standing a foot or two away from the window.

Design Patterns: History

- **Low Sill Pattern**
  illustrates a couple of important characteristics of patterns:
  - Design (and more generally engineering) is about balancing conflicting forces or constraints
  - Design patterns provide general solutions at a medium level of abstraction
  - Patterns aren't dogma

Why Design Patterns?

- **Designing object-oriented code is hard, and designing *reusable* object-oriented software is even harder [E. Gamma]**
  - Design patterns foster development of reusable and extensible software
- **Patterns enable programmers to "…recognize a problem and immediately determine the solution without having to stop and analyze the problem first."**
  - Design patterns capture and document design experience
- **Patterns provide a framework/vocabulary for communicating about object oriented design at a high level of abstraction**

Properties

- **Patterns do...**
  - provide common vocabulary for communicating about design
  - help document software architectures
  - capture essential parts of a design in compact form (solution schema)
  - describe software abstractions
  - document design experience

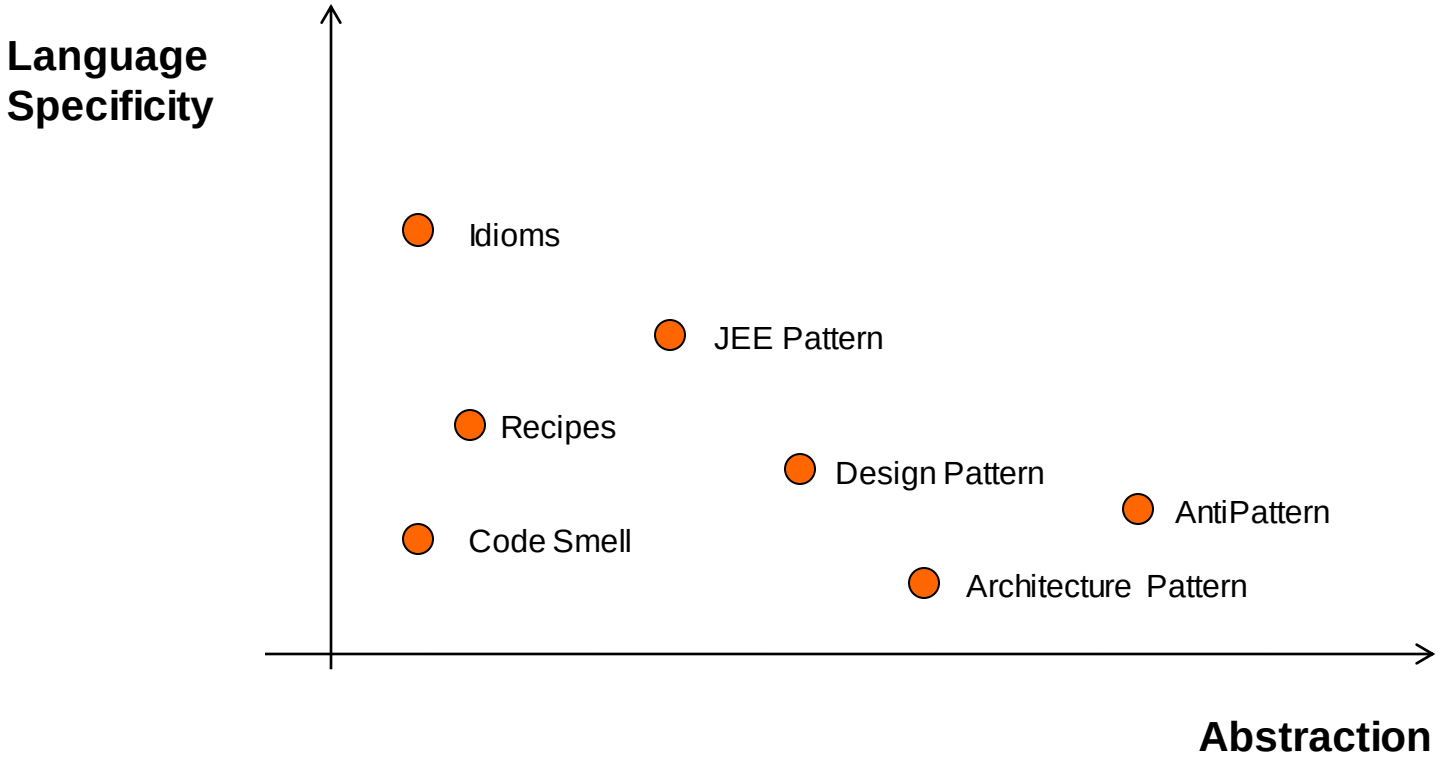
- **Patterns do not...**
  - provide an exact solution, only provide a solution schema
  - solve all design problems
  - necessarily be object-oriented
  - replace a good software designer / architect – but support them

Types of Patterns

- **Software Patterns**
  - Architectural patterns (system design) Pattern-Oriented Software Architecture (POSA)
  - Design patterns (micro architectures) Gang of Four (GOF) Design Patterns
  - Coding Patterns Idioms / Recipes (low level)
- **Analysis Patterns**
  - Recurring & reusable analysis models used in requirements engineering
- **Domain-Specific patterns**
  - UI patterns, security patterns
- **Anti-Patterns => Refactoring**

Types of Software Patterns

- **Classification**



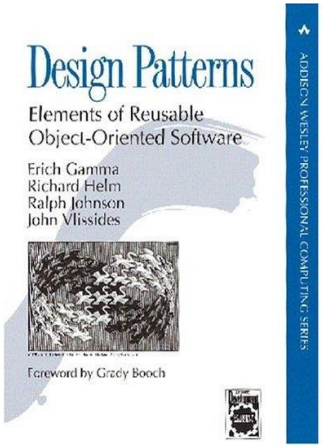
Pattern Form: GoF

- **Pattern name and classification**
- **Intent** what does pattern do / when the solution works
- **Also known as** other known names of pattern (if any)
- **Motivation** the design problem
- **Applicability** situations where pattern can be applied
- **Structure** a graphical representation of classes in the pattern
- **Participants** the classes participating and their responsibilities
- **Collaborations** of the participants to carry out responsibilities
- **Consequences** trade-offs, concerns
- **Sample code** code fragment showing possible implementation
- **Implementation** hints, techniques
- **Known uses** patterns found in real systems
- **Related patterns** closely related patterns

References

- **Head First Design Patterns**
  Eric Freeman, Elisabeth Freeman and Kathy Sierra
  O'Reilly, 2015
  ISBN 978-0596007126

- **Design Patterns: Elements of Reusable Object-Oriented Software**
  Erich Gamma, Richard Helm, Ralph Johnson,
  John Vlissides,
  Prentice Hall
  ISBN 978-0201633610

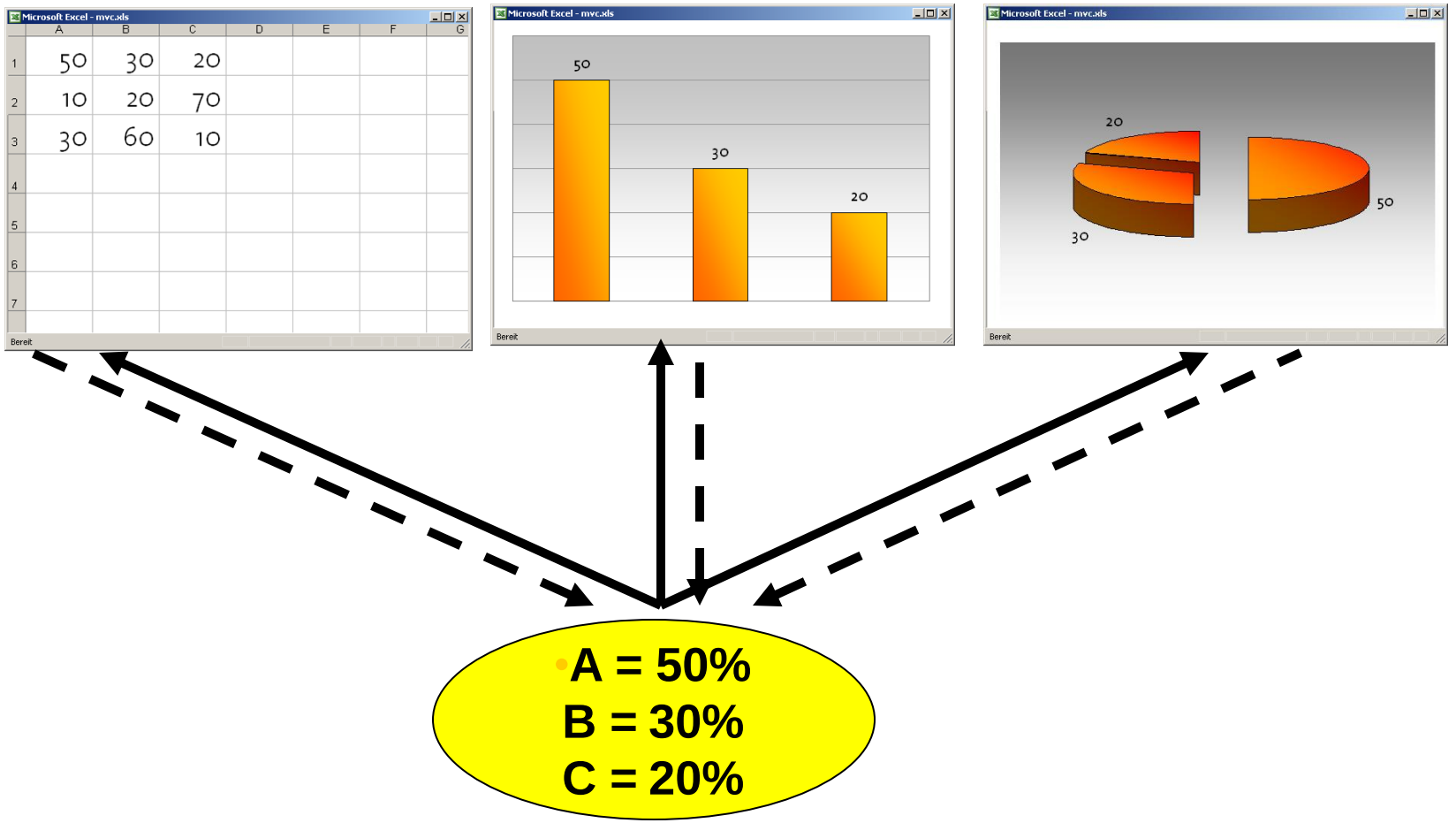


Observer Pattern

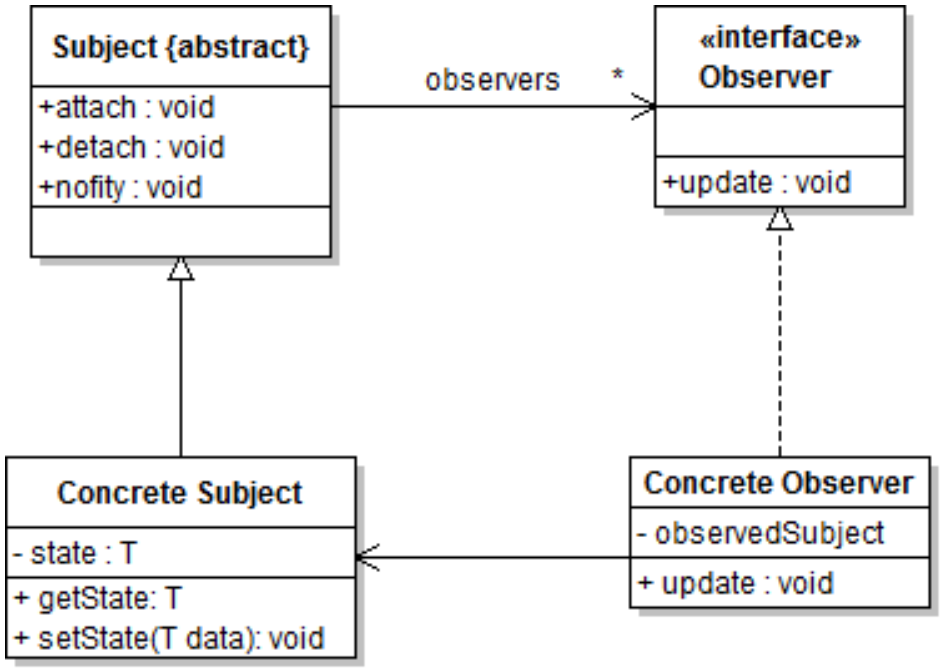
- **Intent**
  - One-to-many relation between objects which allows to inform the dependent objects about state changes
  - Consistency assurance between cooperating objects without connecting them too much
  - Notification of a dependent object without knowing it

- **Also Known As**
  - Publish-Subscribe
  - Listener Pattern

Observer Pattern: Motivation



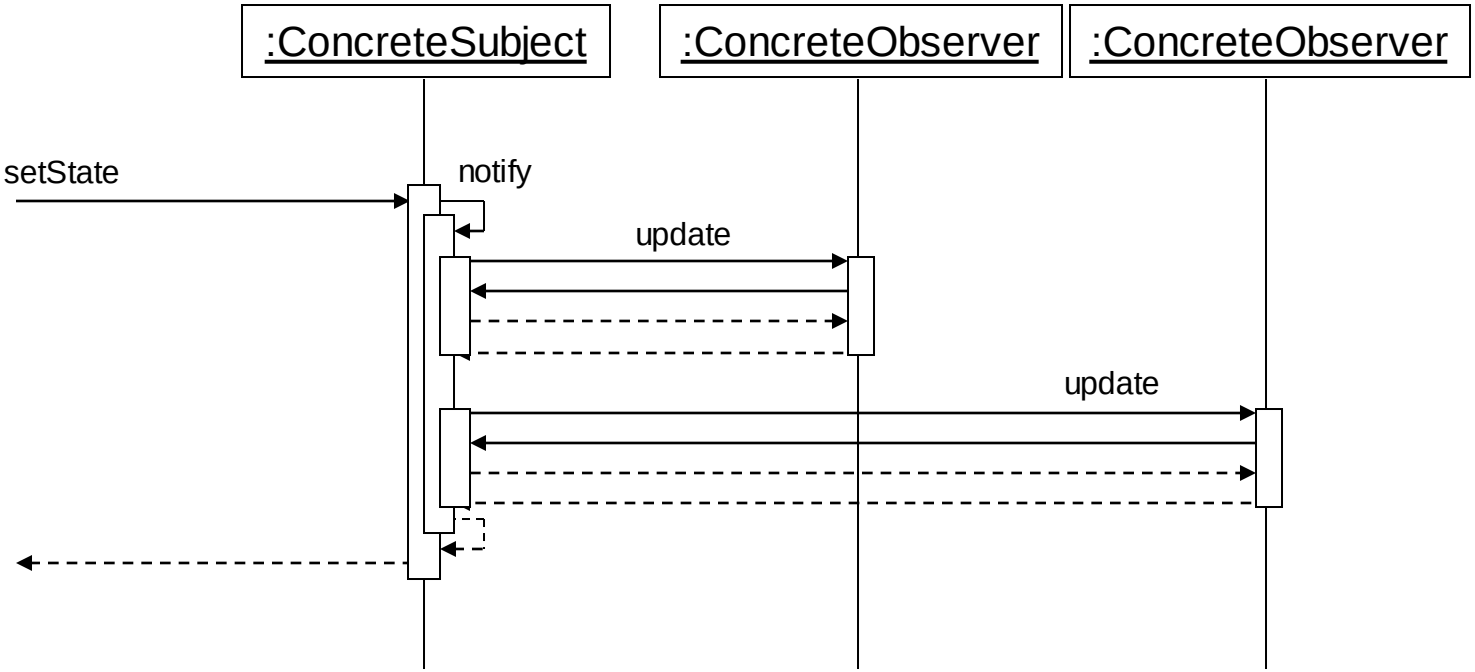
Observer Pattern: Structure



Observer Pattern: Participants

- **Subject** *(may be an abstract class)*
  - Knows its observers only through the Observer interface
  - Offers methods to attach or detach an observer
- **Observer** *(is typically an interface)*
  - Defines interface to publish notifications
- **Concrete Subject**
  - Stores state of interest to concrete observer objects
  - Notifies its observers when its state changes
- **Concrete Observer**
  - Implements Observer interface to keep its state consistent with the subject
  - May maintain a reference to a concrete subject

Observer Pattern: Collaborations



Observer Pattern: Consequences

- **Decoupling**
  - Subject only knows Observer interface, no concrete observers
  - Update = dynamically bound => dynamic invocation
  - Subject and observer may belong to different abstraction layers
- **Support for broadcast communication**
  - Notification is broadcast, subject does not care about number of observers
  - It is up to the observer to handle or ignore notifications
- **Unexpected updates**
  - A simple operation on a subject may cause a cascade of updates

Observer Pattern: Sample Code (1/2)

```
interface Observer {
   void update();
}

abstract class Observable {
   private final List<Observer> observers = new ArrayList<>();

   public void addObserver(Observer o) { observers.add(o); }
   public void removeObserver(Observer o) { observers.remove(o); }

   protected void notifyObservers() {
      for(Observer obs : observers) {
         obs.update();
      }
   }
}
```

Observer Pattern: Sample Code (2/2)

```
class Sensor extends Observable {
   private int temp;
   public  int  getTemperature() { return temp; }
   public  void setTemperature(int val) {
      temp = val;
      notifyObservers();
   }
}

class SensorObserver implements Observer {
   private final Sensor s;
   SensorObserver (Sensor s) { this.s = s; s.addObserver(this); }

   public void update() {
      System.out.println("Sensor has changed, new temperature is "
                                 + s.getTemperature());
   }
}
```

Observer Pattern: Known Uses

- **AWT Event handling**
  - ActionListener, MouseListener, MouseMotionListener, …
- **Swing: Model-View separation**
  - JTable
  - JList
- **Java Beans Event Notifications**

```
public void addPropertyChangeListener(PropertyChangeListener)
public void removePropertyChangeListener(PropertyChangeListener)

public void addVetoableChangeListener(VetoableChangeListener)
public void removeVetoableChangeListener(VetoableChangeListener)
```

- **JavaFX Data Binding**