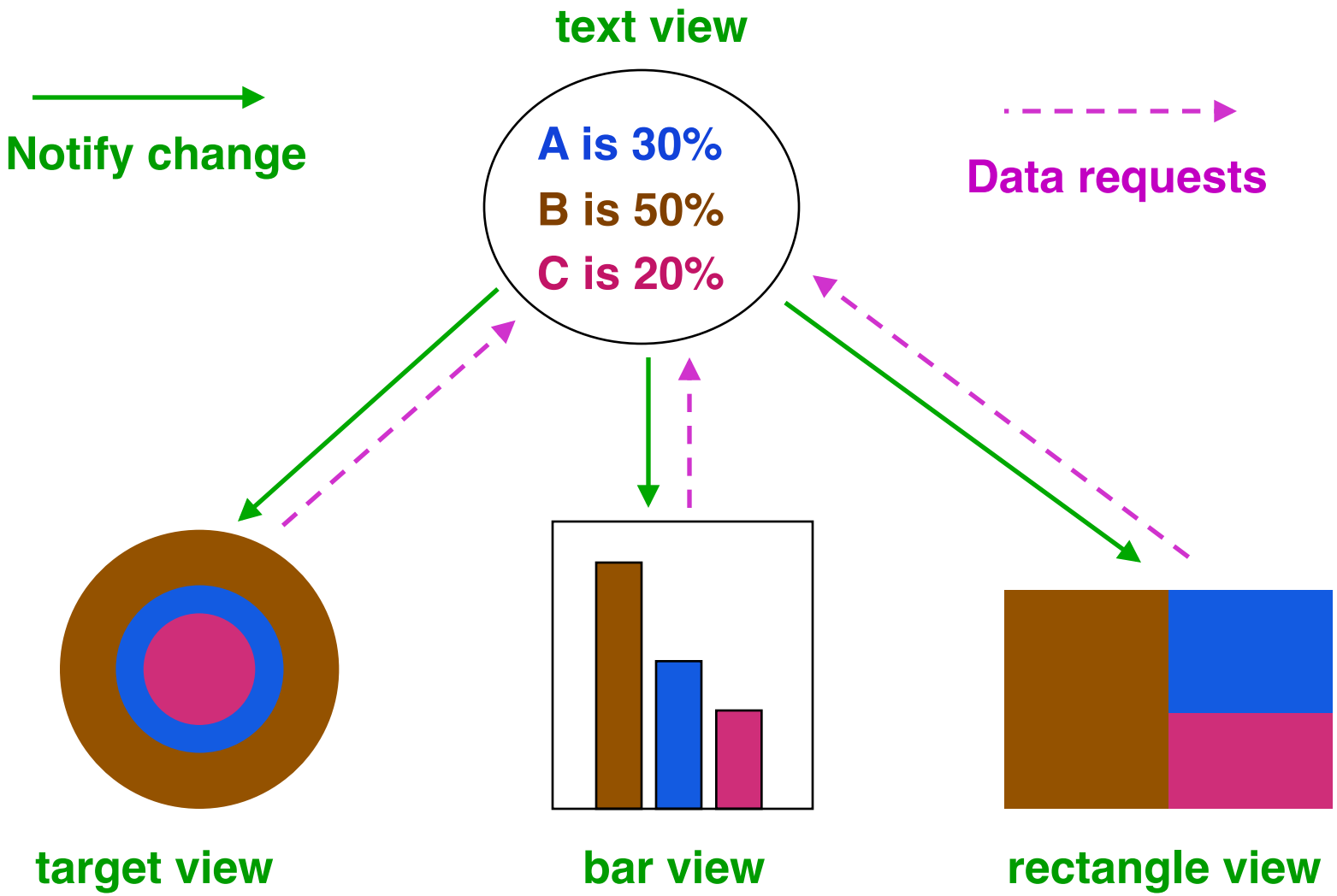
Observer Pattern – Behavioural

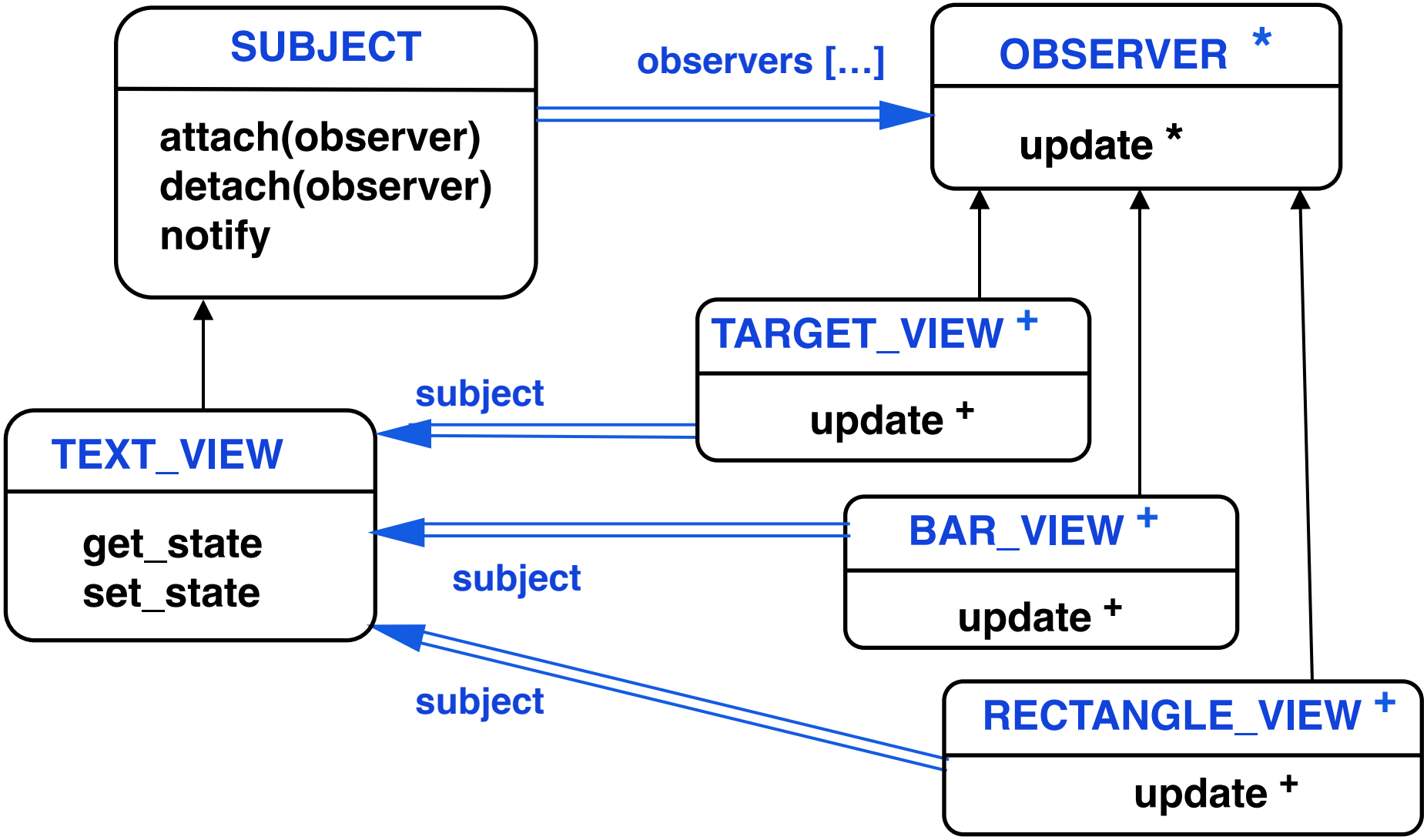
- Intent
  - » **Define one-to-many dependency**
    - > **When one subject changes state, all observers (dependents) are notified and correspondingly updated**

- Also known as
  - » **Dependents and Publish-Subscribe**

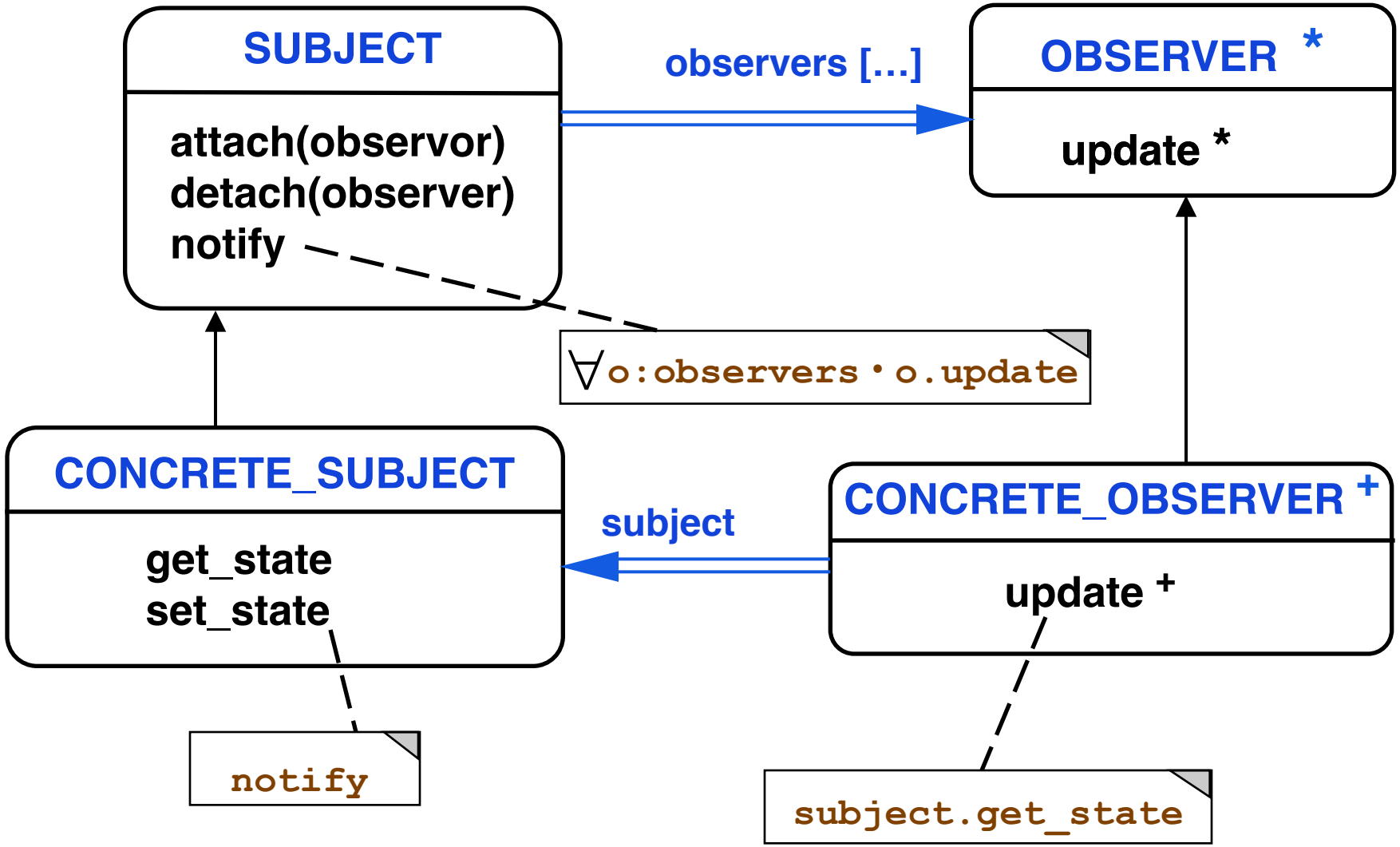
Motivation



Example Architecture



Abstract Architecture

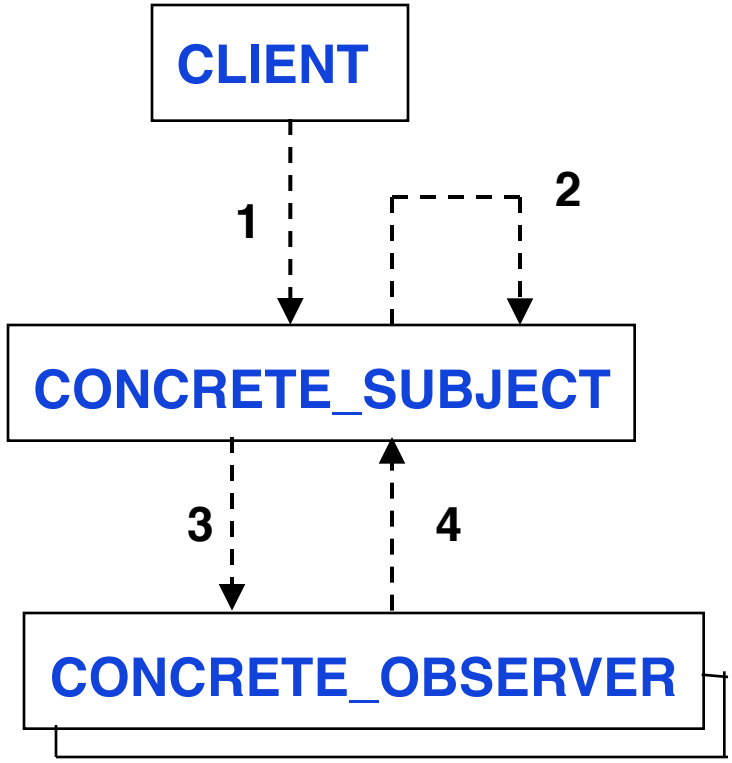


Scenario

- Concrete subject updates all observers, when state is changed by a client

Scenario: Update observers

1 set_state
2 notify
3 update
4 get_state



Participants

- Subject
  - » **Knows its observers**
  - » **Provides interface for attaching, detaching and notifying its observers**

- Observer
  - » **Defines an updating interface for observers**

Participants – 2

- Concrete subject
  - » **Stores state of interest to concrete observers**
  - » **Notifies observers when state changes**

- Concrete observer
  - » **Maintains a reference to its concrete subject**
  - » **Stores state that corresponds to the state of the subject**
  - » **Implements Observer updating interface**

Applicability

- When an abstraction has two aspects, one dependent upon the other
  - » **Encapsulating each aspect as a separate object means you can change and use them independently**
- When changing one object requires changing an indeterminate number of corresponding objects
- When an object needs to notify other objects without making detailed assumptions about those objects, to reduce coupling

Consequences

- Abstract coupling between subject and observer
  - » **Permits changing number of observers dynamically**
  - » **Subject and observer can belong to different layers**
    - > **If they are in one class, then the object spans system layers, which can compromise abstraction by layering**
- Supports broadcast communication
- Can have observers depend upon more than one subject

Consequences – 2

- Observers may also change the state
  - » **Can be expensive as observers are unaware of each other**

- Need additional protocol to indicate what changed
  - » **Can have spurious updates**
    - > **Not all observers participate in all changes**
  - » **Can have clients notify, instead of subject, as clients understand better when updates are needed**
    - > **Leads to errors as clients can forget to update**

Consequences – 3

- Dangling references when subject is deleted
  - » **Notify observers when subject is deleted**
    - > **Cannot delete observers as other subjects may depend upon them**

- Update only when subject state is consistent with respect to observer
  - » **Could be violated when subclasses invoke inherited operations**

Related Patterns

- Mediator pattern is used for change managers
  - » **Change manager mediates between subjects and observers by encapsulating complex update methods**

- Singleton pattern is can be used to make a change manager unique and globally accessible

Observer in Java API

- The class Observer is a direct implementation of the pattern as discussed in these slides