Design Patterns for Java GUIs

Composite, Observer, Command

Design Patterns and Java GUIs

- GUIs (Graphical User Interfaces) in Java are mostly developed using two graphical packages:
  - Swing (graphical objects)
  - AWT (event control)
- Both packages heavily rely on three main Design Patterns:
  - Composite (structural)
  - Observer (behavioral)
  - Command (behavioral)

Structural & Behavioral Patterns

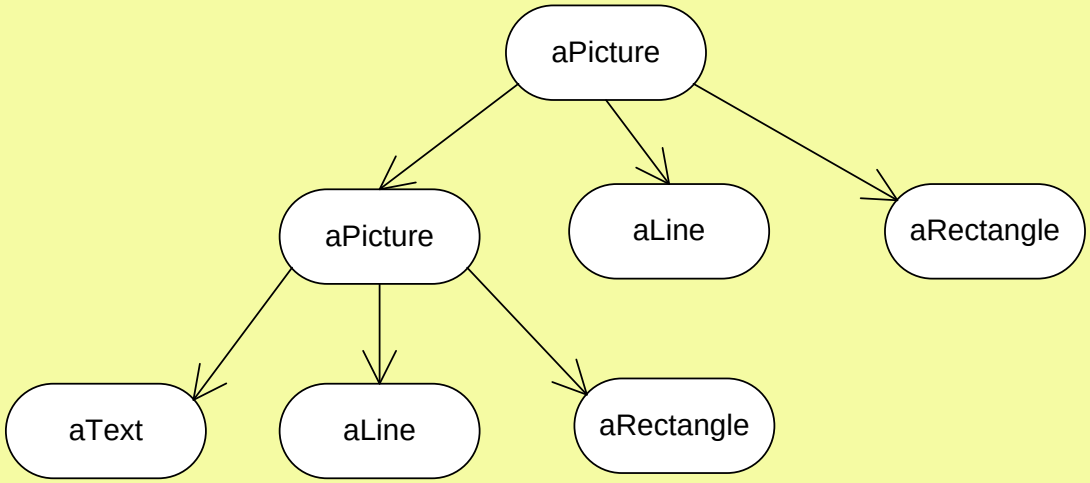
- Structural patterns are concerned with how classes and objects are composed to form larger structures.
- Behavioral patterns are concerned with how classes and objects communicate with each other and assignment of responsibilities among objects (algorithms)

Composite Pattern: Motivation

- Suppose we want to design a graphic editor which creates complex graphical objects
- Objects can be:
  - primitive basic objects such as lines, polygons and text;
  - container objects obtained composing other objects
- We want to model both primitive and container objects as instances of the same class

Composite Pattern: Motivation (2)

- A Graphic object has a tree-like recursive structure:
  - inner nodes are containers made of other graphical objects
  - terminal nodes (or leaves) are primitive shapes (lines, rectangles, text, etc.)



Composite Pattern: Motivation (3)

- In general, we want to be able to perform the following operations on a node:
  - draw it
  - add a child node (graphical object)
  - remove a child node (graphical object)
  - access one of its child nodes

Composite Pattern: Basic Idea

Problem:

Code must treat primitive and container objects differently, even if users most of the time treat them identically

Composite Pattern idea:

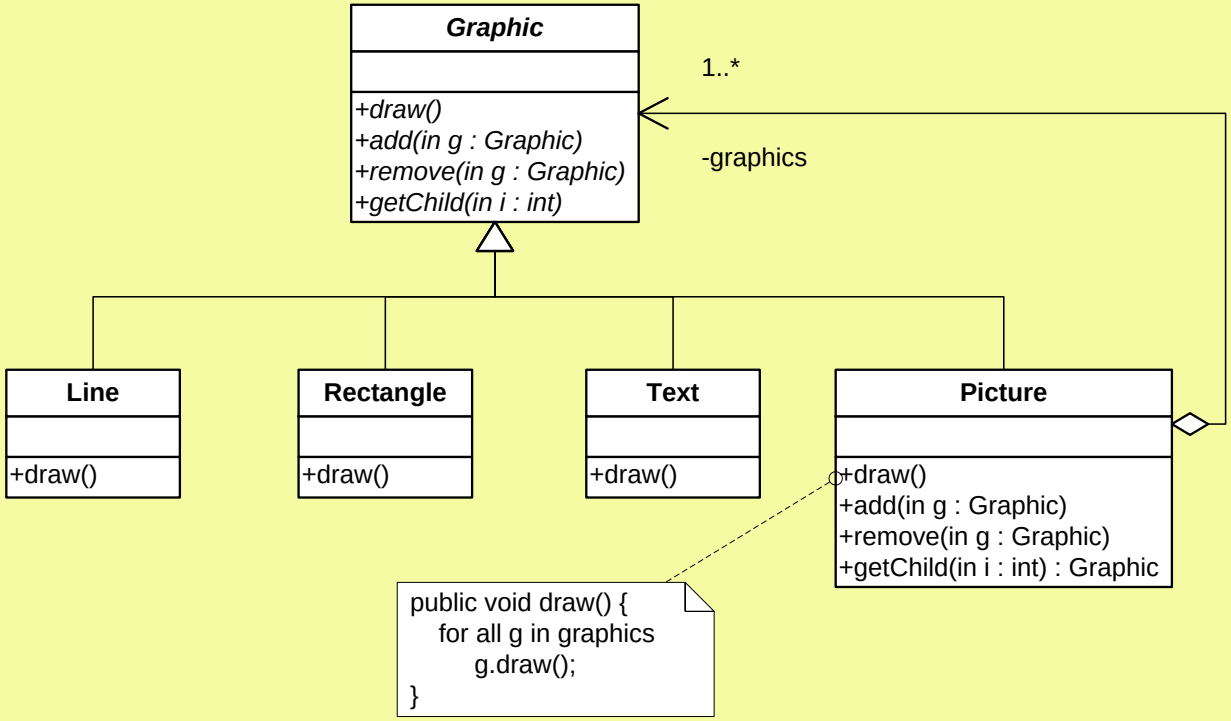
Create an abstract class which represents both primitive and container objects

Composite Pattern: Intent

Intent

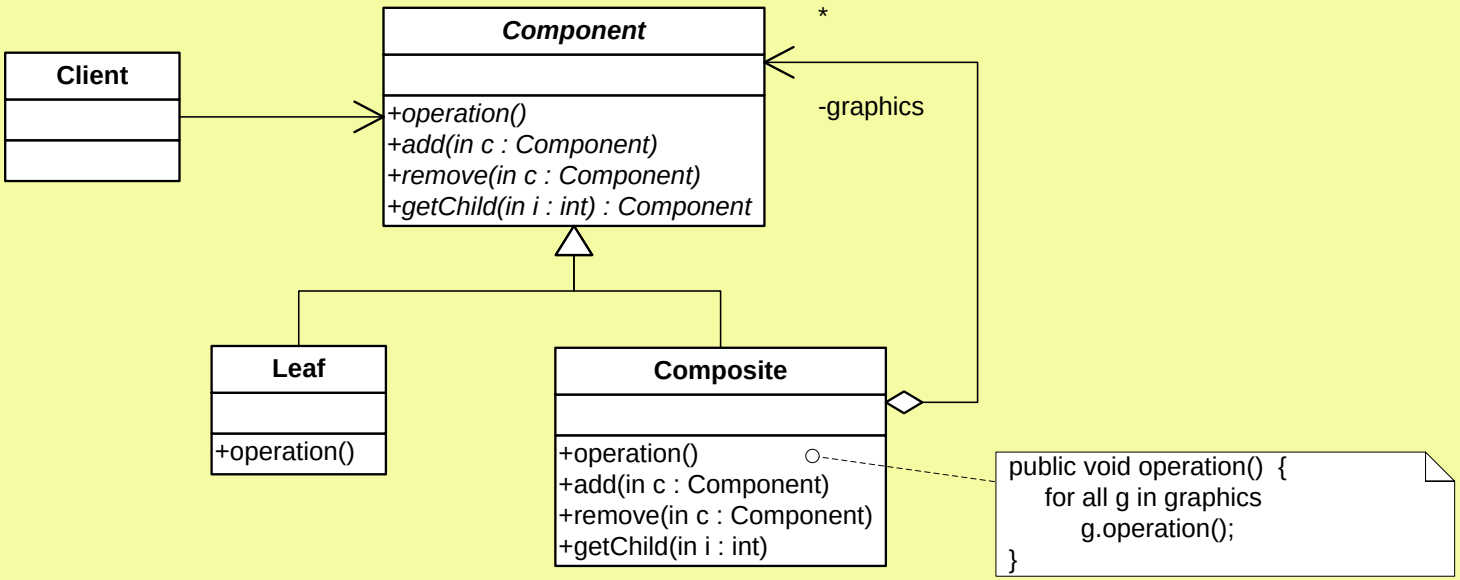
- compose objects into tree structures to represent part-whole hierarchies
- let clients treat individual objects and compositions of objects uniformly

Composite Pattern: Motivation



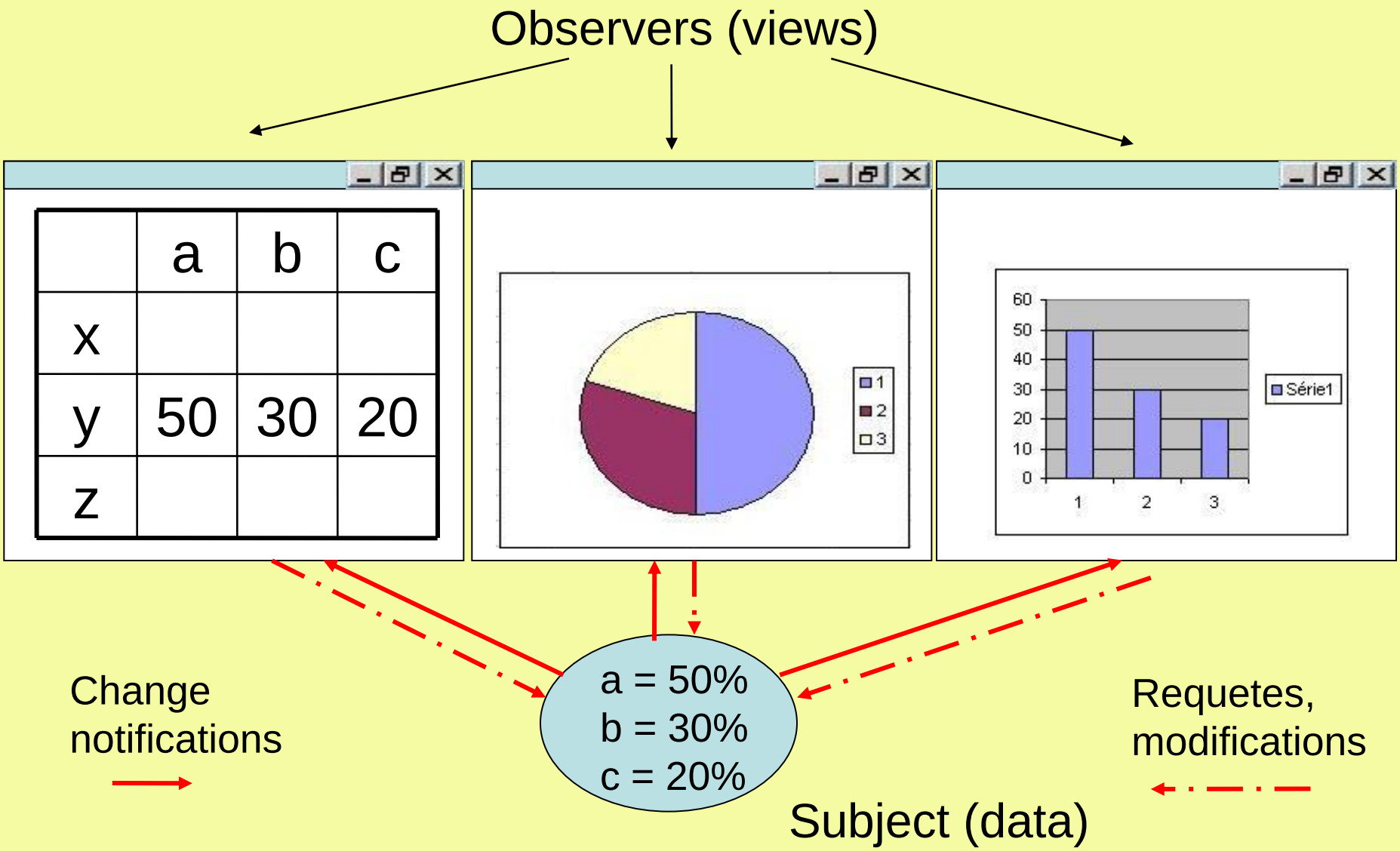
- Graphic: abstract class to manipulate
  - primitive objects (lines, rectangles, text, etc.)
  - container objects
- Picture: defines the container-type class of objects

Composite Pattern: Structure



- All nodes inherit from an abstract Component class which:
  - declares an abstract operation, common to all subclasses
  - declares container-specific operations, which will be implemented by leaves with empty code
- Every primitive object implements operation()

Pattern Observer: Motivation



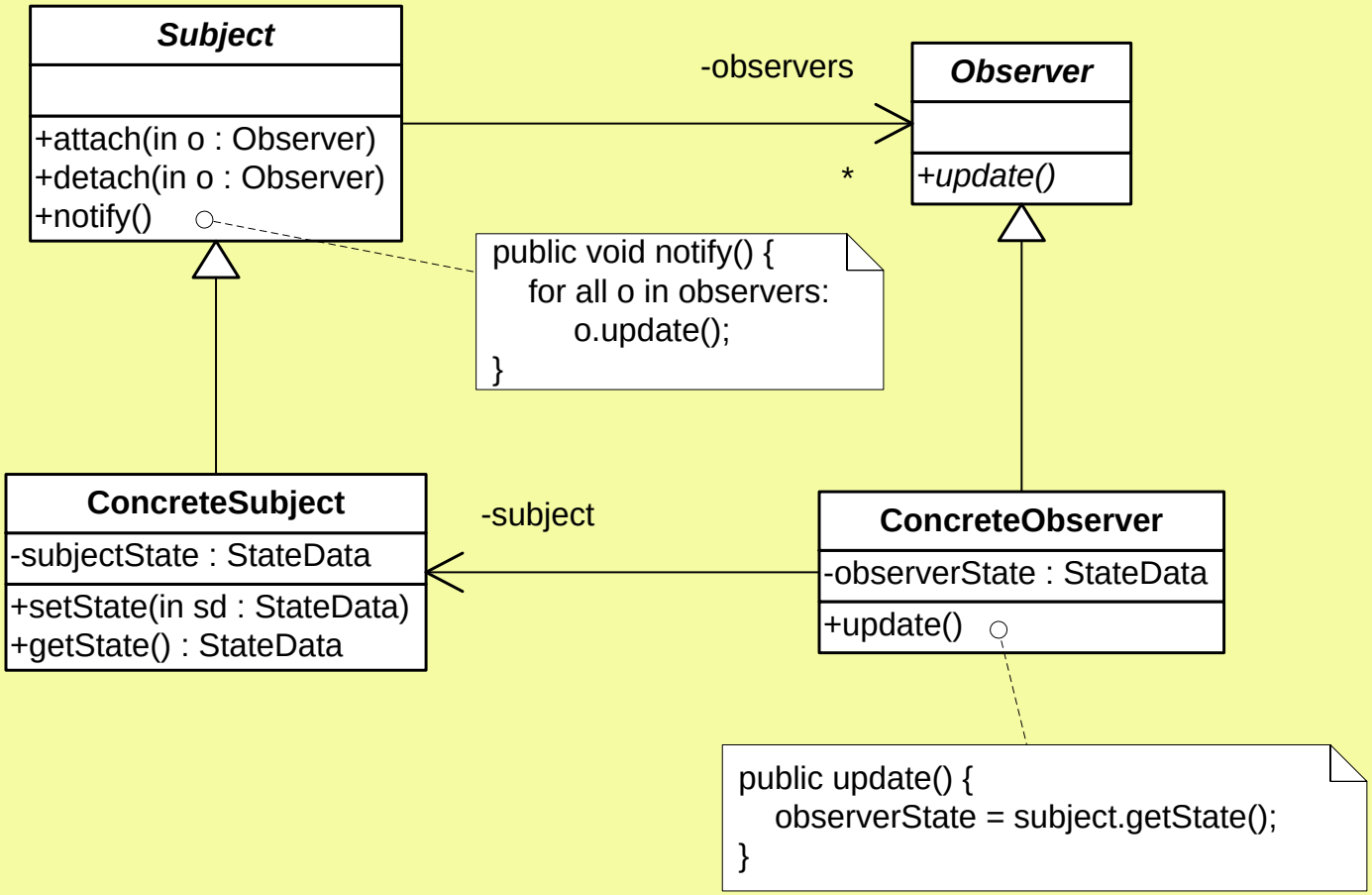
Pattern Observer: Motivation(2)

- Several GUI toolkits separates the presentational aspects of the user interface (views) from the underlying application data
- It is necessary to maintain consistency between related views but
- view classes must not be too tightly coupled, because that reduces reusability

Observer: intent

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

Observer: Structure & Participants



Observer: Participants

Subject

- knows its observers, any number of Observer objects may observe it
- provides an interface for attaching and detaching Observer objects

Observer

- defines an update interface for objects that should be notified of changes

ConcreteSubject

- stores states of interest to ConcreteObserver objects
- sends a notification to its observers when its state changes

ConcreteObserver

- maintains a reference to a ConcreteSubject object
- stores states that should stay consistent with the subject's

Observer: Behavior