Design Patterns & Software Architecture

Decorator

dr. Joost Schalken-Pinkster

Windesheim University of Applied Science

The Netherlands

The contents of these course slides is (in great part) based on:
Chris Loftus, *Course on Design Patterns & Software Architecture for NEU*. Aberystwyth University, 2013.
Jeroen Weber & Christian Köppe, *Course on Patterns and Frameworks*. Hogeschool Utrecht, 2013.
Leo Pruijt, *Course on Software Architecture*. Hogeschool Utrecht, 2010-2013.

Session overview

- Decorator

Decorator Design Pattern

Let’s find a design pattern

Will now present, on the board,
and using Eclipse, a solution that
utilizes the decorator design pattern.

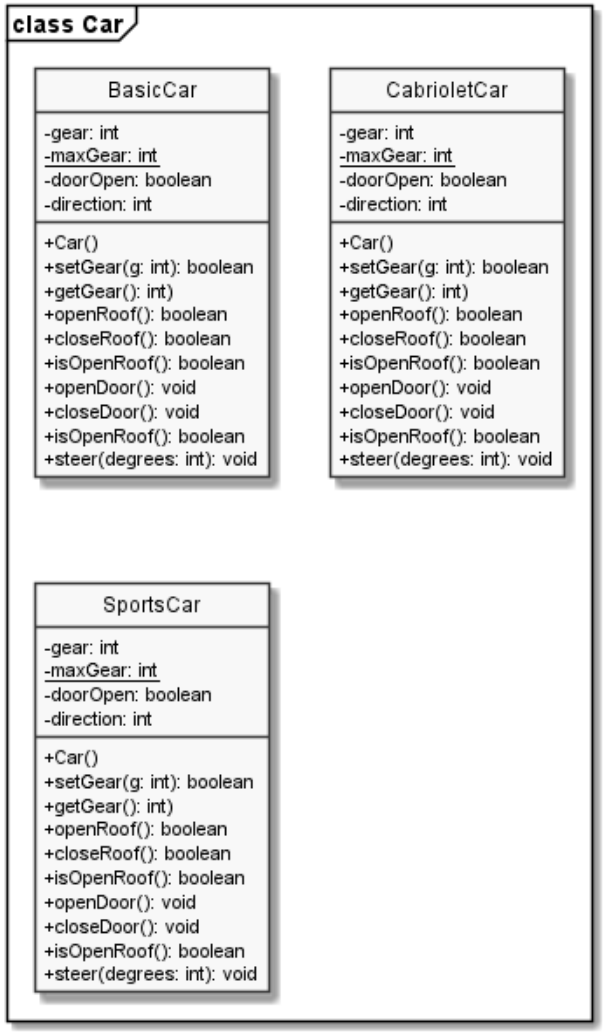
Case: Car system Requirements

Write a program that can model the behaviour of a Car:

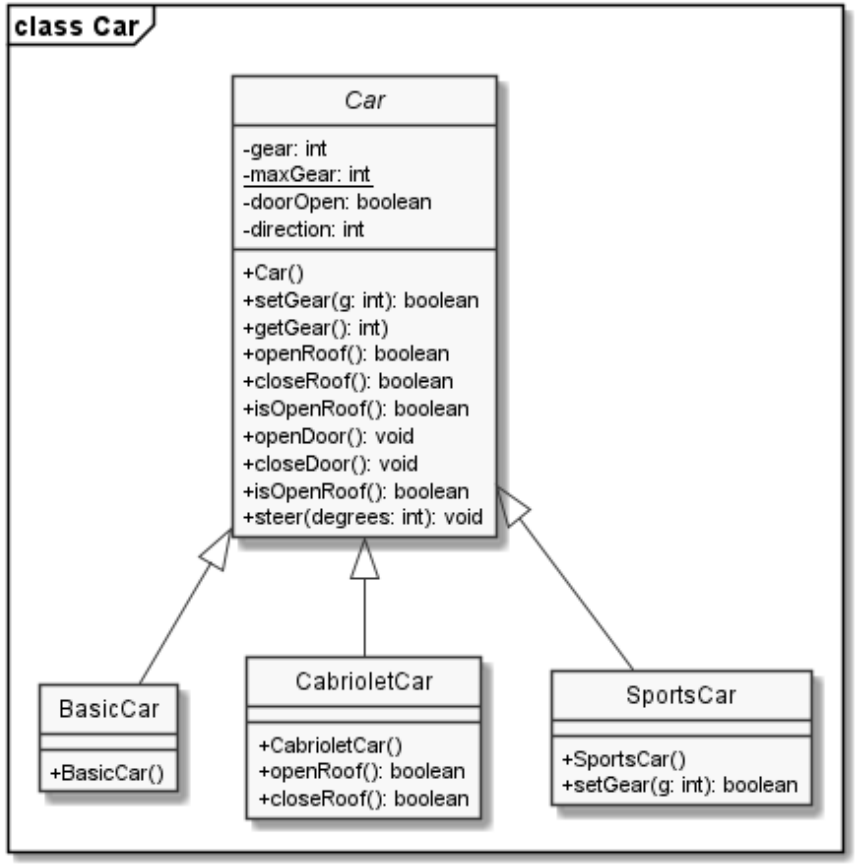
- There are three types of Cars: basic cars, cabriolet (can open roof) and sports car (has 6 gears instead of 5).
- Cars have methods for setting speed, gear, opening doors, opening roof (if possible) and steering.

- New requirement: add a police sports car
- New requirement: add a basic police car
- New requirement: add a car with a maximum speed of 120 km/hour.

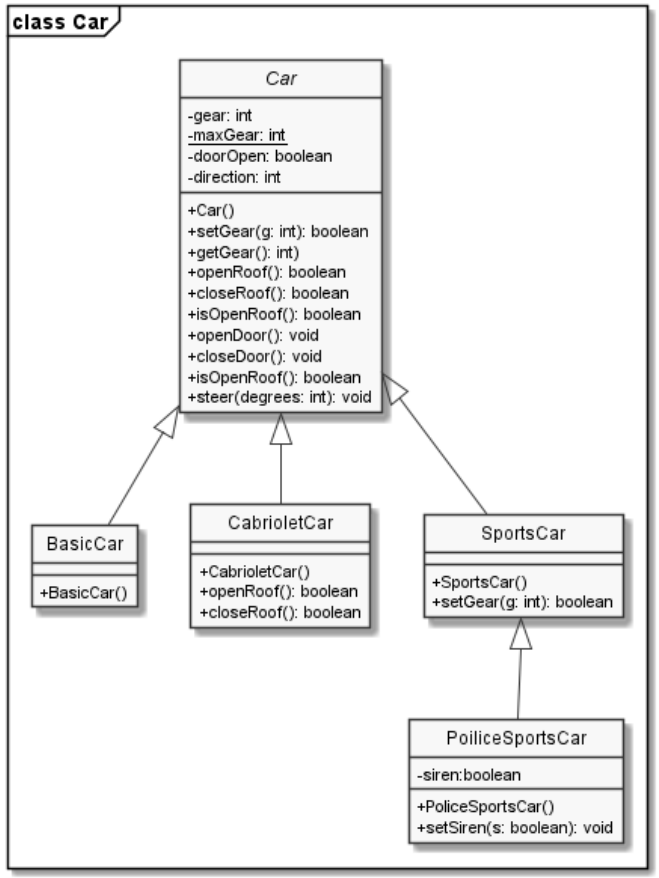
Case: Car system
Design 1: Initial requirements



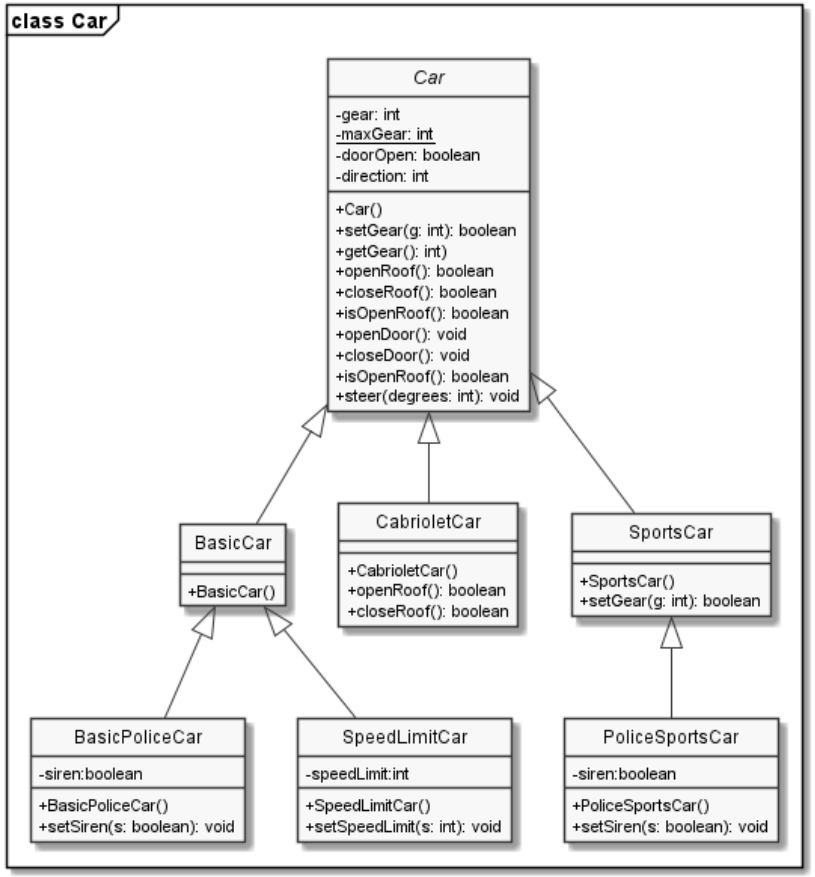
Case: Car system
Design 2: Initial requirements



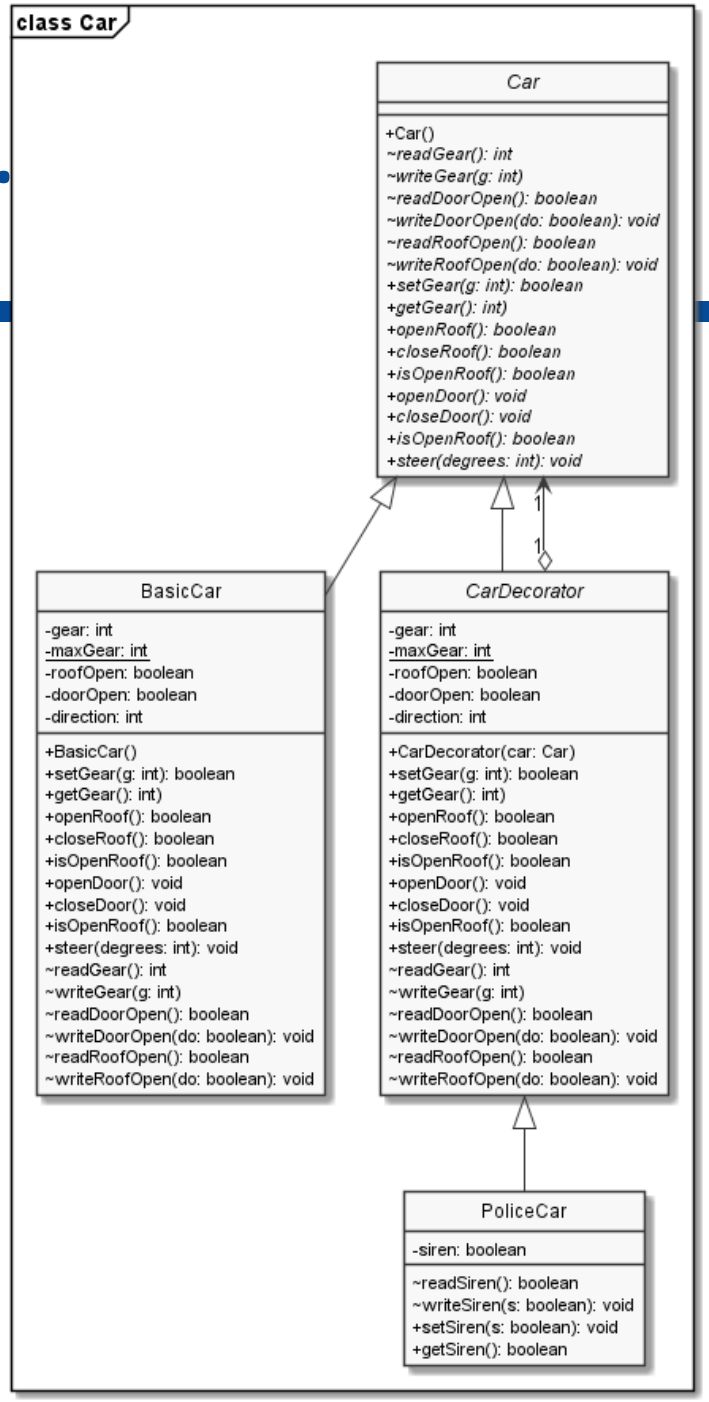
Case: Car system
Design 3: With police car



Case: Car system
Design 4: With police car, basic car, cabriolet…
One big mess…



Case: Car system
Design 5: With police car, basic car, cabriolet..
Decorator



Design Principle:
Open closed principle

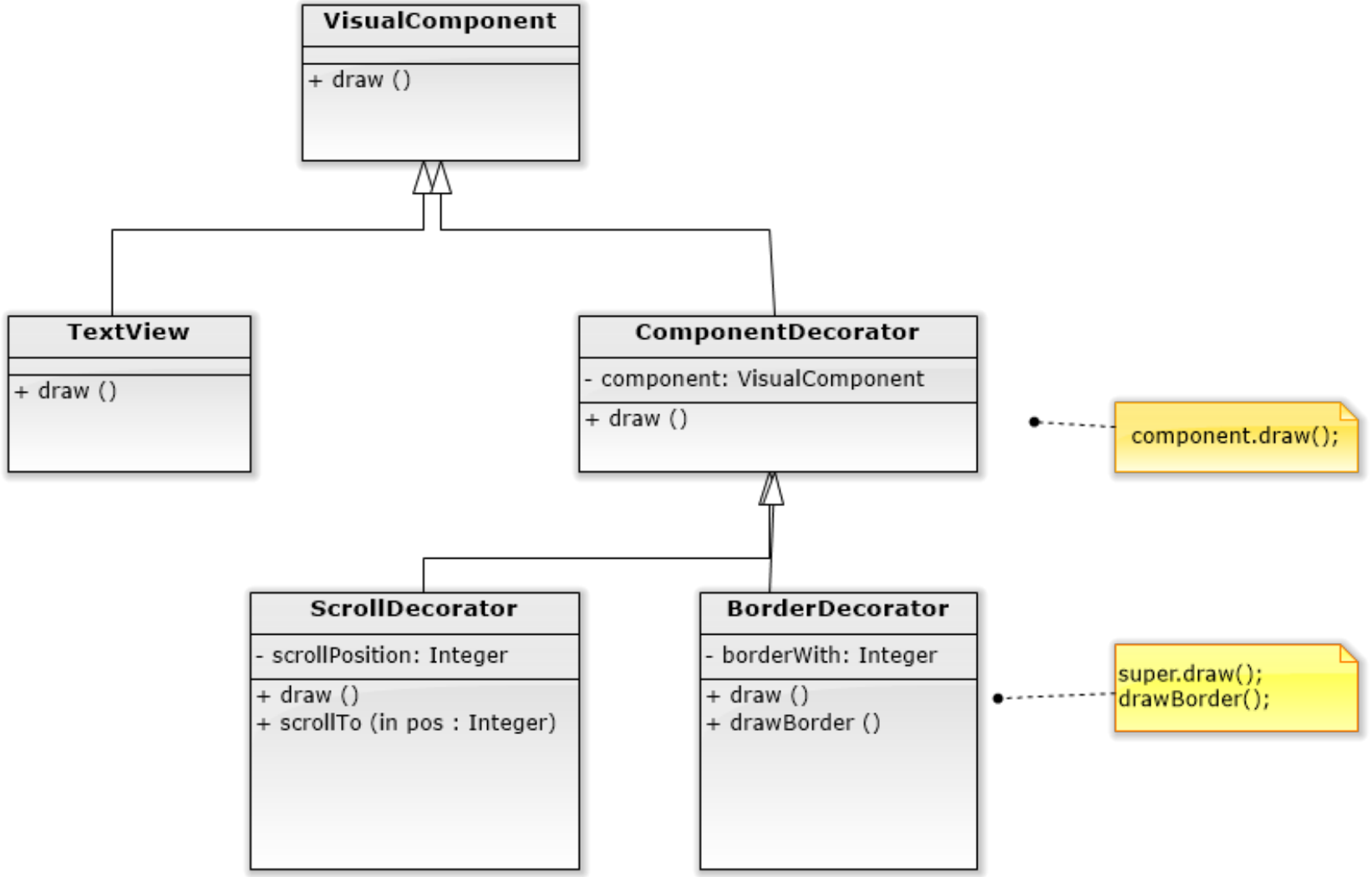
Classes should be open for extension, but closed for modification.

E. Freeman, E. Freeman, B. Bates, K. Sierra, Head First Design Patterns. O'Reilly, 2004. p. 86

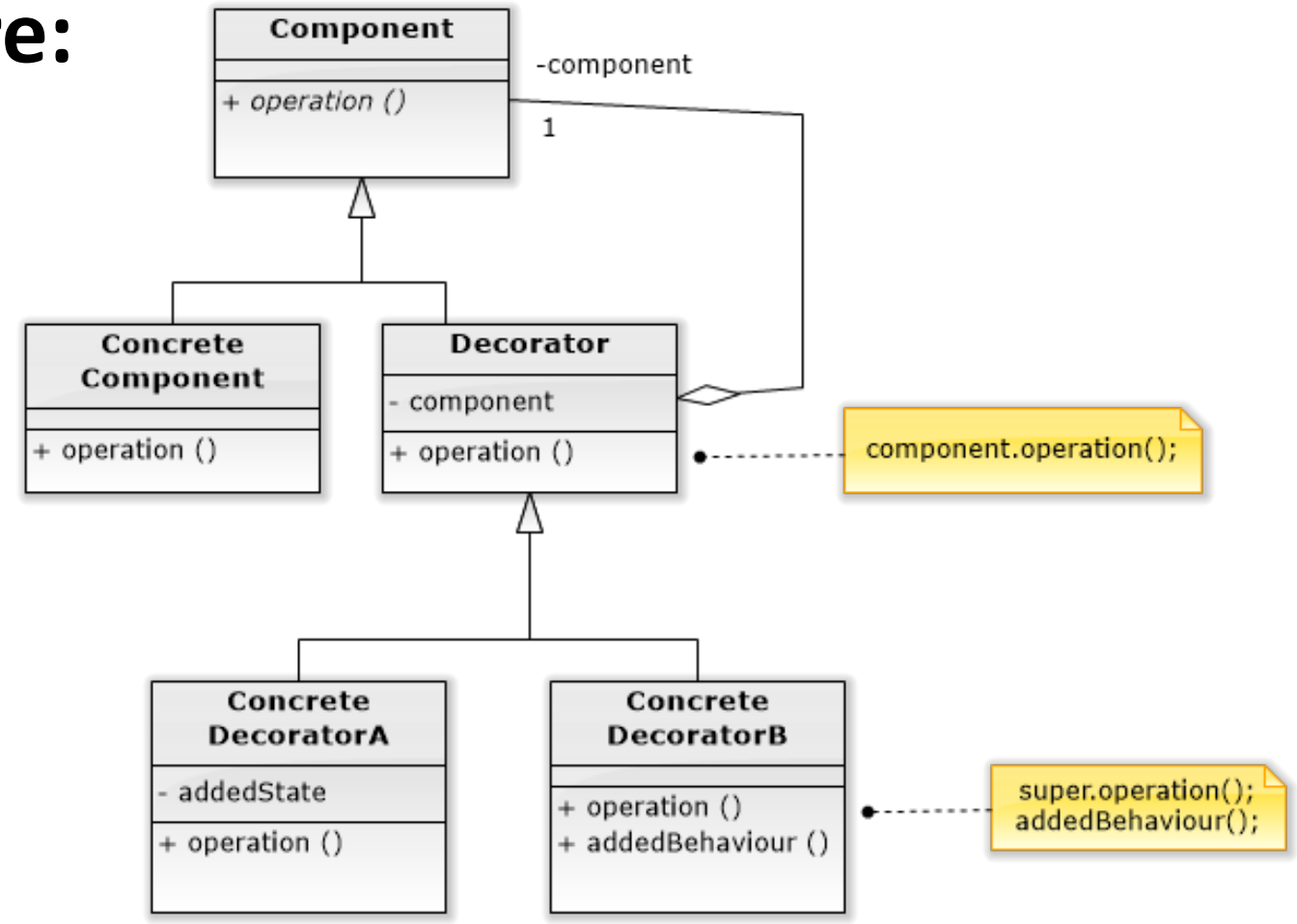
Decorator pattern definition

- **Intent**: Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality…

- **Motivation**: Sometimes we want to add responsibilities to single objects, and not to an entire class. A graphical user interface toolkit, for example, should let you add properties like borders or behaviour, e.g. scrolling to any GUI component…


VisualComponent
+ draw ()
TextView
+ draw ()
ComponentDecorator
- component: VisualComponent
+ draw ()
component.draw();
ScrollDecorator
- scrollPosition: Integer
+ draw ()
+ scrollTo (in pos : Integer)
BorderDecorator
- borderWith: Integer
+ draw ()
+ drawBorder ()
super.draw();
drawBorder();

- **Structure:**



Gamma (1995), Design Patterns: Elements of Reusable Object-Oriented Software, p. 199

- **Participants**:
  - Component (VisualComponent):
    - Defines the interface for objects that can have responsibilities added to them dynamically…
  - ConcreteComponent (TextView):
    - Defines an object to which additional responsibilities can be added…
  - Decorator:
    - Maintains a reference to a Component object and subclasses/implements Component …
  - ConcreteDecorator (e.g. BorderDecorator):
    - Adds responsibilities to the Component…

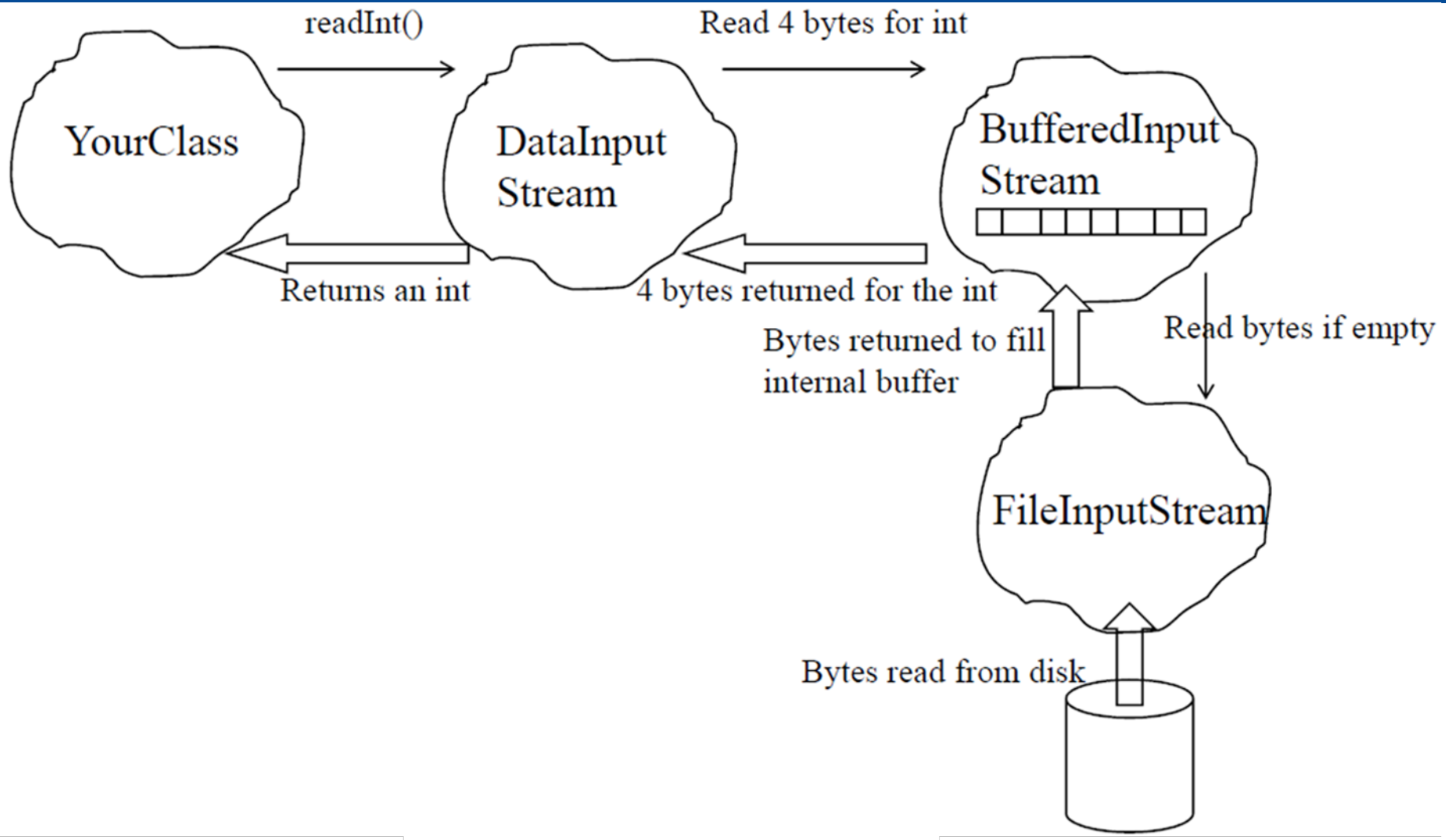
- **Collaborations**:
  - Decorator forwards requests for its component object…

- **Consequences**:
  - More flexibility than static inheritance…
  - Avoids feature-laden classes high up in the hierarchy…
  - A decorator and its component aren't identical…
  - Lots of little objects…

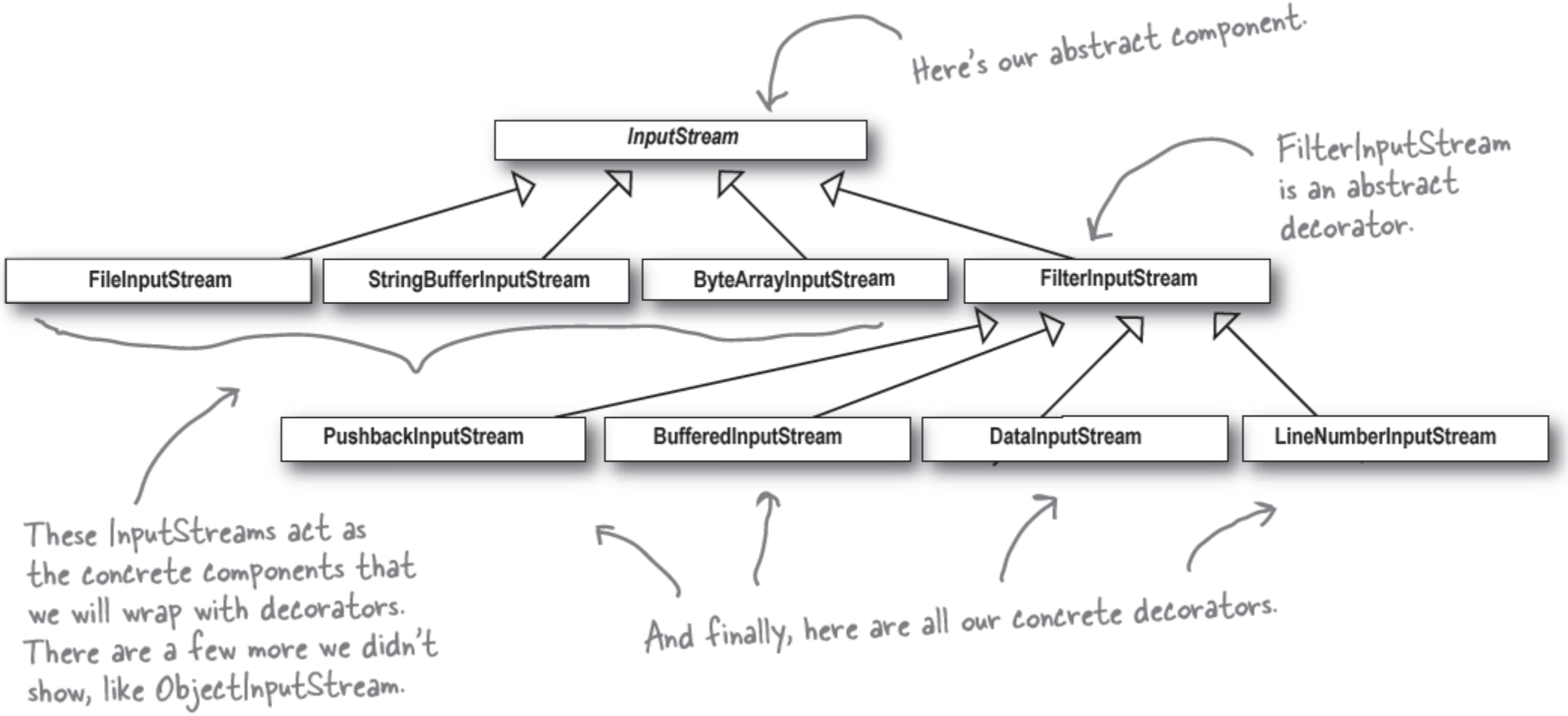
Gamma (1995), Design Patterns: Elements of Reusable Object-Oriented Software, p. 199-200

- **Implementation**: Issues to consider
  - Interface conformance…
  - Omitting the abstract Decorator class…
  - Keeping Component classes lightweight…
  - Change only the skin of the object, not the gut (else use Strategy)

Another example: Filtering Data using Java I/O classes



Another example: Filtering Data using Java I/O classes



- **Applicability**: Use decorator
  - To add responsibility to individual objects dynamically and transparently, that is, without affecting other objects…
  - For responsibilities that can be withdrawn…
  - When extension by subclass is impractical…

Reading

For this lesson please read:

- Chapter 3 (Decorator Objects) of Head First Design Patterns