



الجامعة السورية الخاصة
SYRIAN PRIVATE UNIVERSITY

المحاضرة 5

كلية الهندسة المعلوماتية

مقرر تصميم نظم البرمجيات

Design Patterns: State Pattern, Decorator

د. رياض سنبل

Design Patterns

■ **Creational Patterns**

Factory Method
Builder

(abstracting the object-instantiation process)

Abstract Factory
Prototype

Singleton

■ **Structural Patterns**

Adapter
Decorator
Proxy

(how objects/classes can be combined)

Bridge
Facade

Composite
Flyweight

■ **Behavioral Patterns**

Command
Mediator
Strategy
Template Method

(communication between objects)

Interpreter
Observer
Chain of Responsibility

Iterator
State
Visitor

State Pattern

The Problem

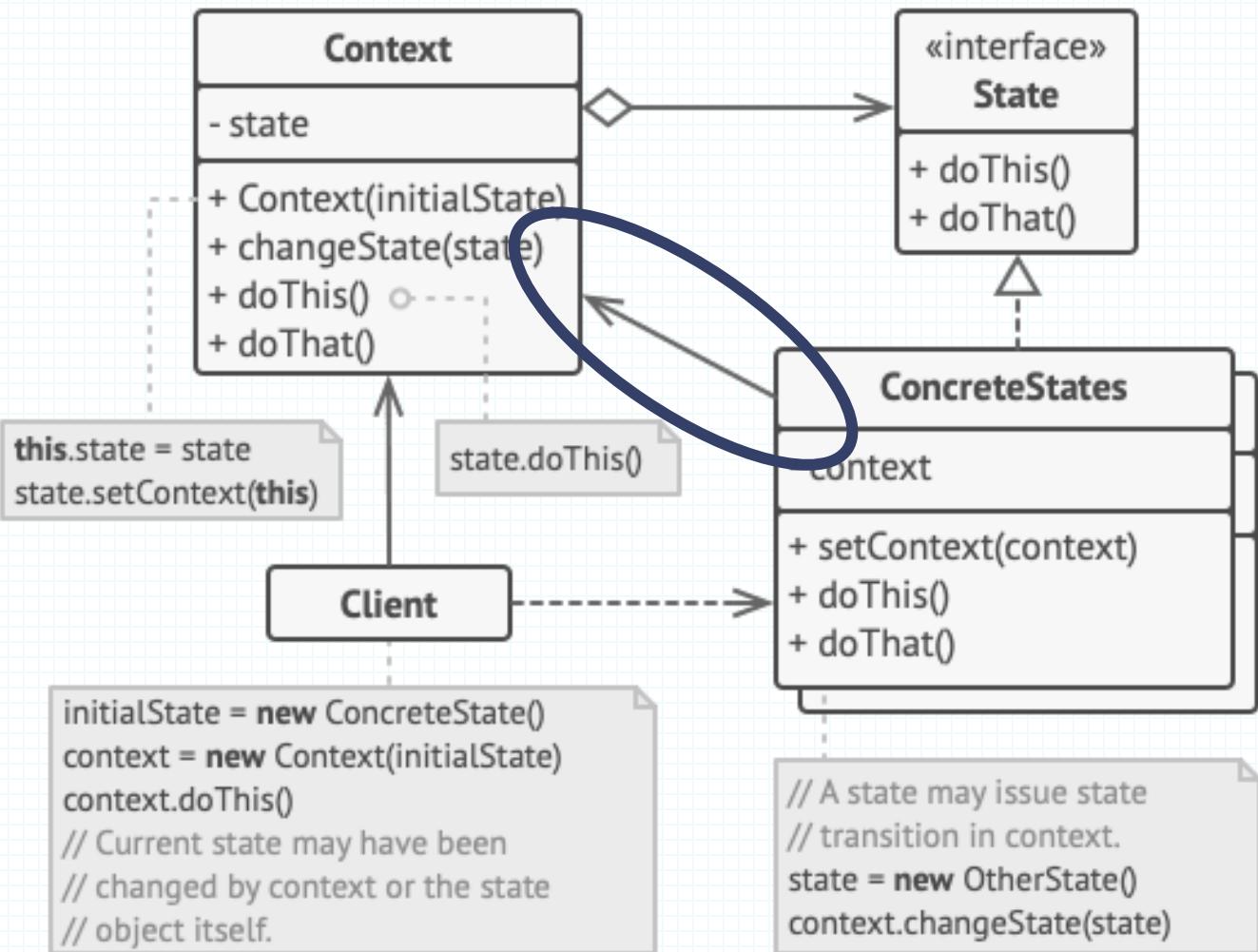
- An object's behavior changes based on its internal state!
- Operations have large, multipart conditional statements that depend on the object's state.
- This state is usually represented by one or more enumerated constants.
- Often, several operations will contain this same conditional structure.
- **The State pattern puts each branch of the conditional in a separate class**

Example

- Imagine that we have a Document class. A document can be in one of three states: Draft, Moderation and Published. The publish method of the document works a little bit differently in each state:
 - In Draft, it moves the document to moderation.
 - In Moderation, it makes the document public, but only if the current user is an administrator.
 - In Published, it doesn't do anything at all.



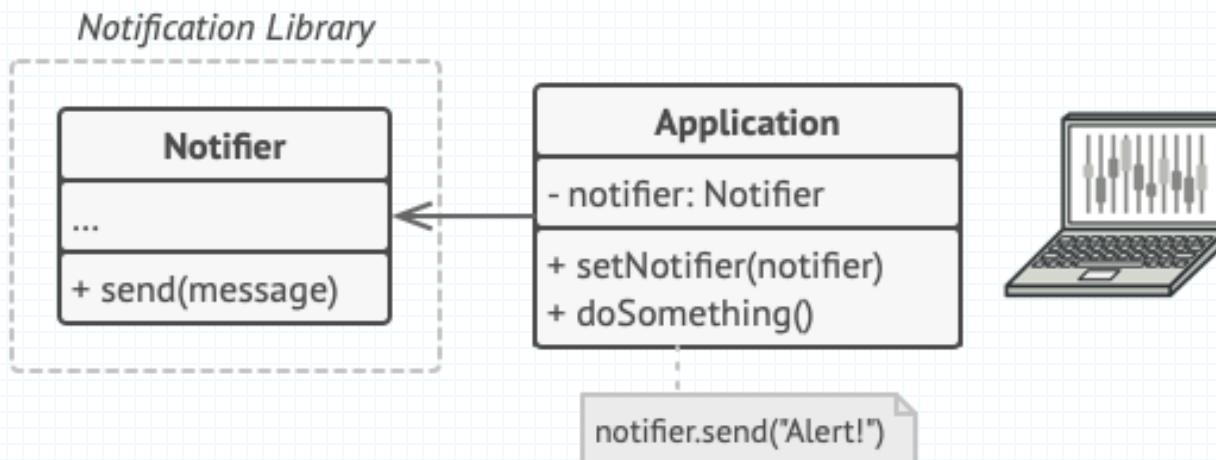
State Pattern



Decorator Pattern

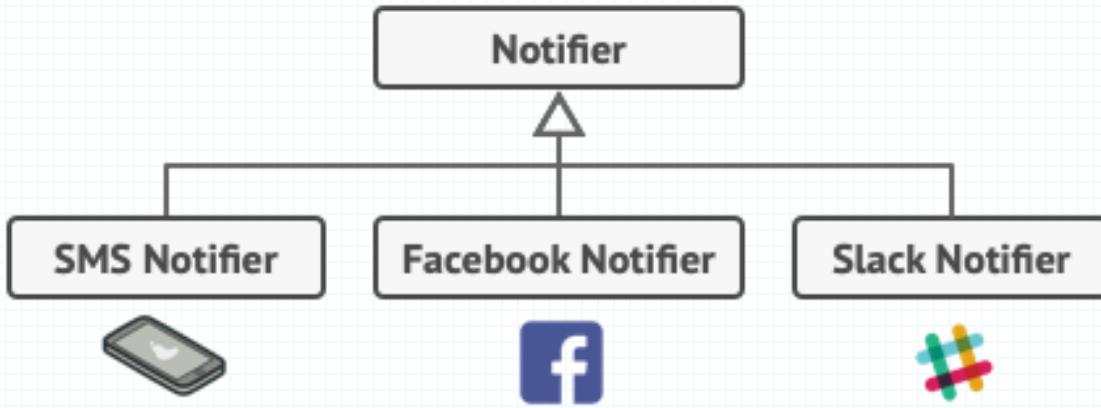
Again.. Let us start with the problem 😊

- you're working on a notification library which lets other programs notify their users about important events.
- The initial version of the library was based on:
 - the *Notifier* class that had only a few fields, a constructor and a single *send* method. The method could accept a message argument from a client and send the message to a list of emails that were passed to the notifier via its constructor.



Req. Changed:

The library expect more than just email notifications. They would like to receive an SMS about critical issues. Others would like to be notified on Facebook and, of course, the corporate users would love to get Slack notifications.

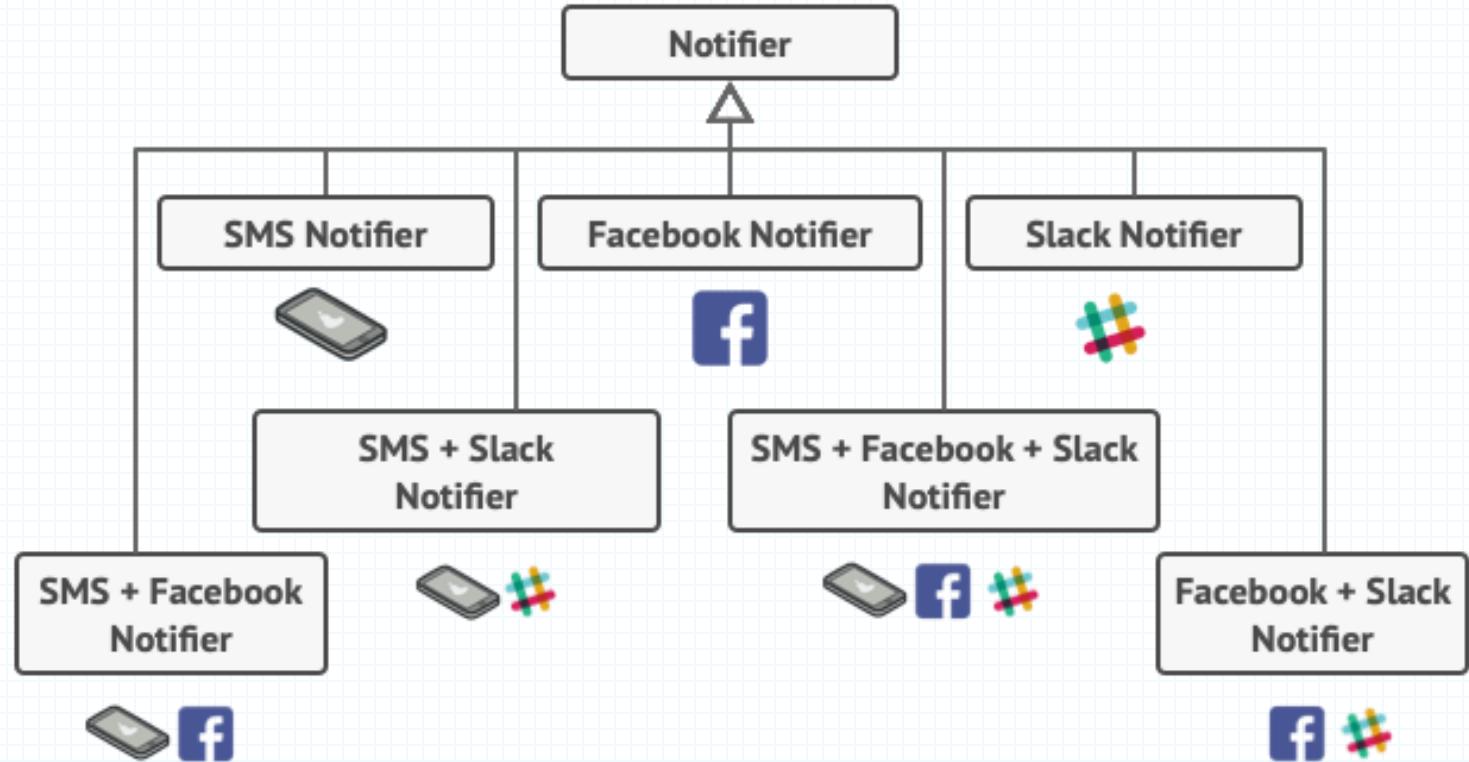


Solution:

- ✓ Extended the Notifier class and put the additional notification methods into new subclasses.
- ✓ Now the client was supposed to instantiate the desired notification class and use it for all further notifications.

New Change 😞

Why can't we use several notification types at once?
If your house is on fire, you'd probably want to be informed through every channel.



BAD Solution

Solution (Decorator)

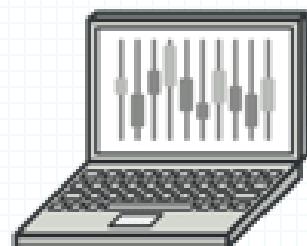
```
stack = new Notifier()
if (facebookEnabled)
    stack = new FacebookDecorator(stack)
if (slackEnabled)
    stack = new SlackDecorator(stack)

app.setNotifier(stack)
```

Application

```
- notifier: Notifier
+ setNotifier(notifier)
+ doSomething()
```

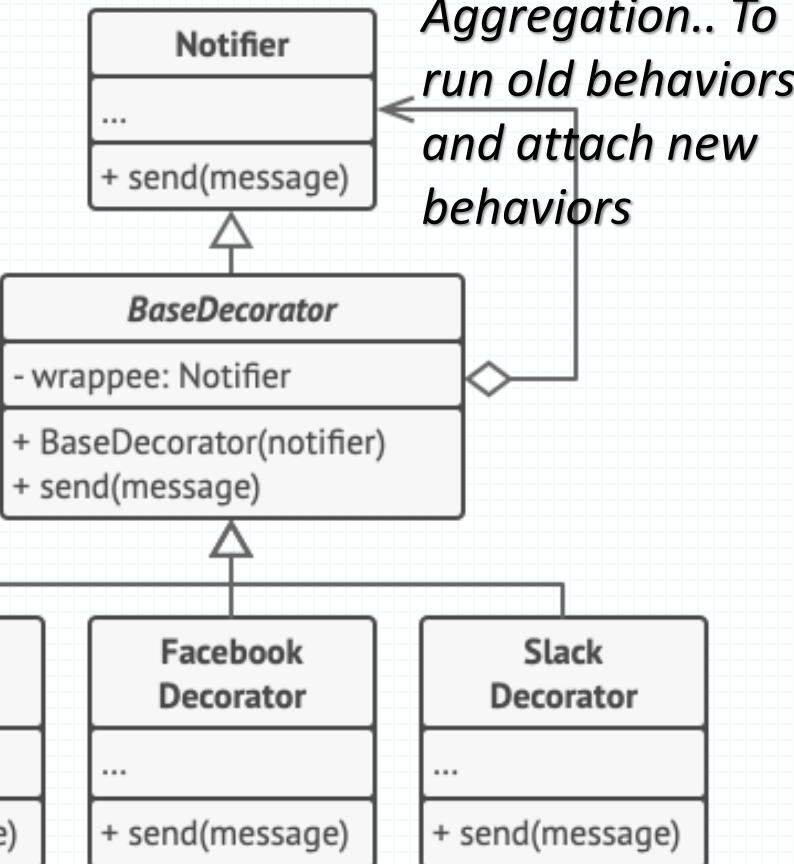
notifier.send("Alert!")
// Email → Facebook → Slack



Inheritance.. So the new instance has the same “type”

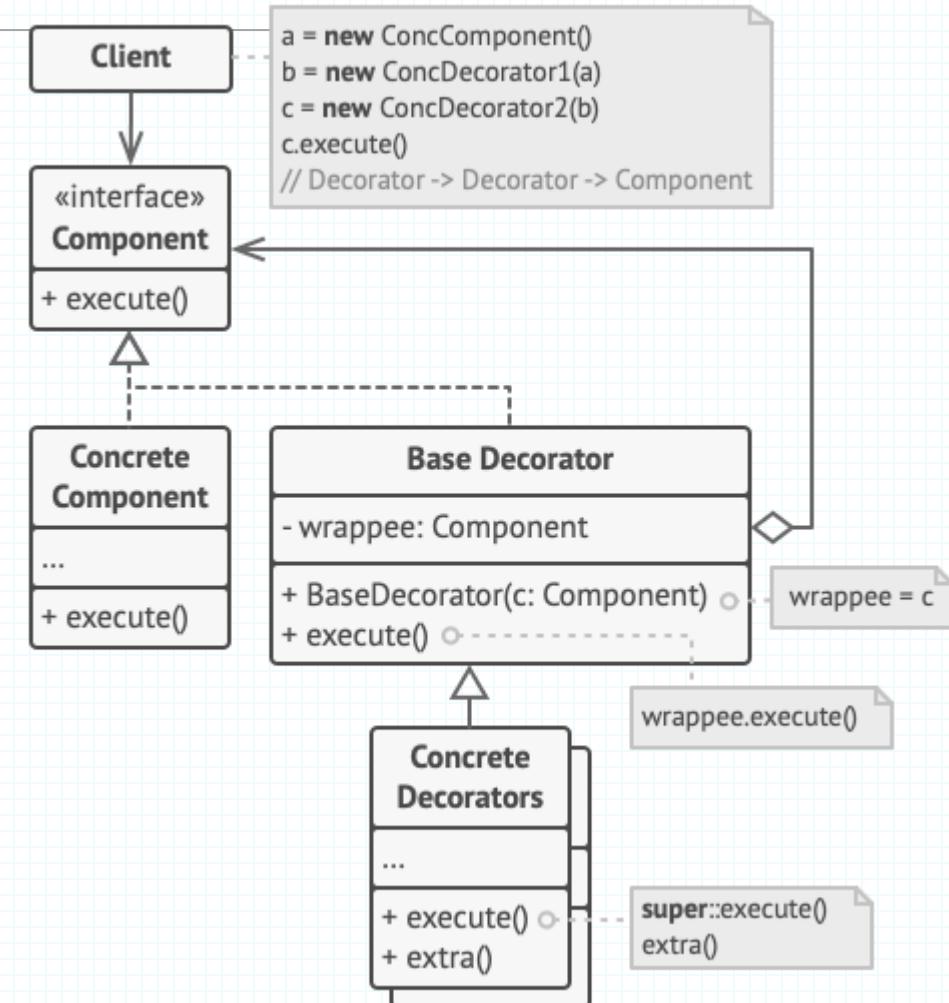
wrappee.send(message);

super::send(message);
sendSMS(message);

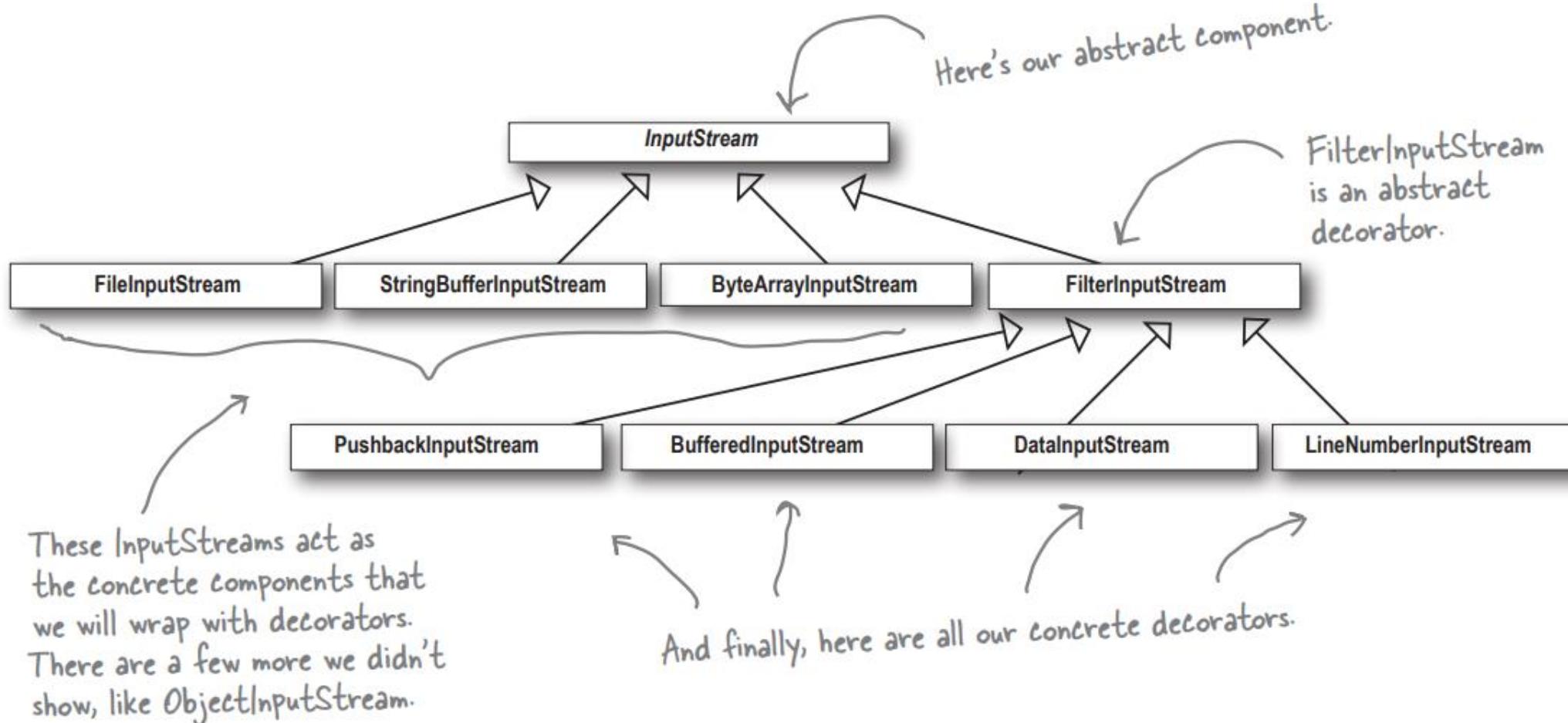


Decorator Pattern

- Decorator is a structural design pattern that lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.
- Use the Decorator pattern when you need to be able to assign extra behaviors to objects at runtime without breaking the code that uses these objects.



Another Example.. From Java.io



Another Example.. From Java.io

