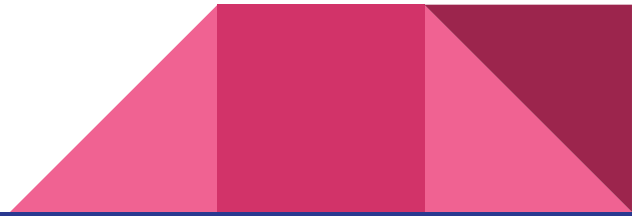
Design Patterns Tutorial

CS 319

10/12/2021

Agenda

1. Overview of Design Patterns
2. Decorator Pattern Exercise
3. Singleton Pattern Exercise
4. About the Lab Session



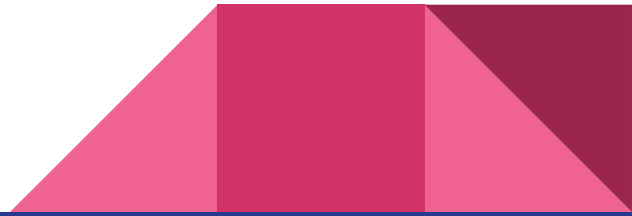
What are design patterns?

v · t · e	Software design patterns	[hide]
Gang of Four patterns	Creational	Abstract factory · Builder · Factory method · Prototype · Singleton
	Structural	Adapter · Bridge · Composite · Decorator · Facade · Flyweight · Proxy
	Behavioral	Chain of responsibility · Command · Interpreter · Iterator · Mediator · Memento · Observer · State · Strategy · Template method · Visitor
Concurrency patterns		Active object · Balking · Binding properties · Double-checked locking · Event-based asynchronous · Guarded suspension · Join · Lock · Monitor · Proactor · Reactor · Read write lock · Scheduler · Thread pool · Thread-local storage
Architectural patterns		Front controller · Interceptor · MVC · ADR · ECS · *n*-tier · Specification · Publish–subscribe · Naked objects · Service locator · Active record · Identity map · Data access object · Data transfer object · Inversion of control · Model 2
Other patterns		Blackboard · Business delegate · Composite entity · Dependency injection · Intercepting filter · Lazy loading · Mock object · Null object · Object pool · Servant · Twin · Type tunnel · Method chaining · Delegation

Our focus is the *object oriented patterns*

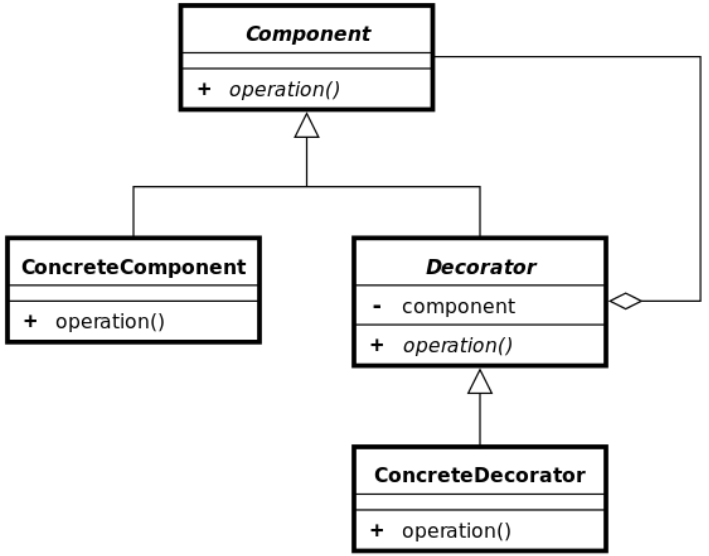
Overview

- Common solutions to common problems
- Flexible and reusable objects to change, test, and reuse easily
- Easier communication between developers
  - Singleton
  - Builder
  - Strategy etc.



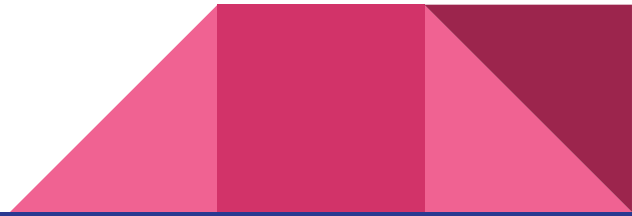
Decorator Pattern

- Add behavior to an individual object dynamically
- Alternative to subclassing



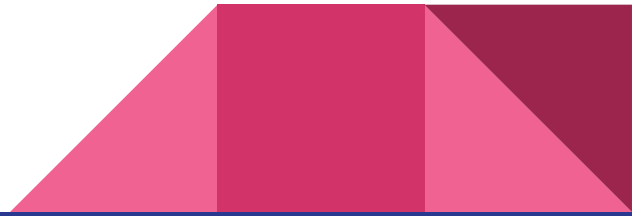
Demo

- Logger class
- How can we extend the Logger’s functionality?
- Inheritance vs Composition



SOLID Principles

- Single responsibility principle
- Open closed principle
- Liskov substitution principle
- Interface segregation principle
- Dependency inversion principle



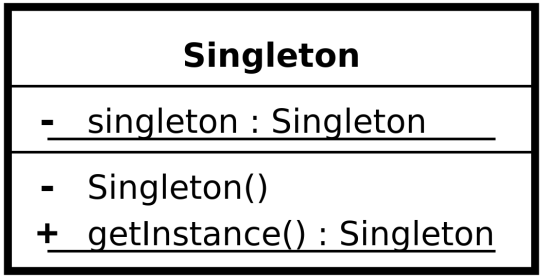
Decorator in Java itself

- Subclasses of *java.io.InputStream*, *OutputStream*, *Reader* and *Writer*

- Also known as Wrapper pattern

Singleton Pattern

- Restricts the instantiation of a class to single instance
- Can be used with other patterns, factory, facade etc.

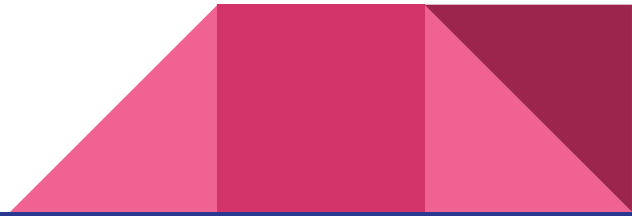


Demo

- Let’s convert the FileLogger to a singleton

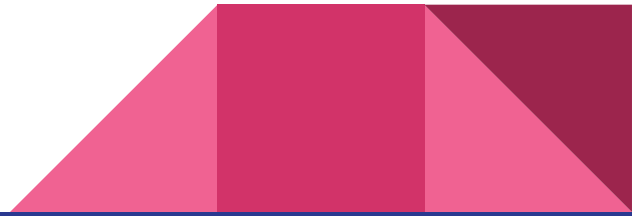
Singleton in Java itself

- java.lang.Runtime *getRuntime()*
- java.awt.Desktop *getDesktop()*
- java.lang.System *getSecurityManager()*



Final Remarks

- In your term project, you are expected to implement a few design patterns
- In a real project, start with a simpler solution and extend it with a pattern when the complexity starts to grow
- Knowing “When to use a pattern” is more important than “How to use a pattern”



Lab Session

- December 17th 18.00-20.00
- Analyze a case
- Find a suitable pattern
- Design the solution
- Implement the solution