



Software Engineering

7. Design Patterns | Thomas Thüm | December 13, 2021

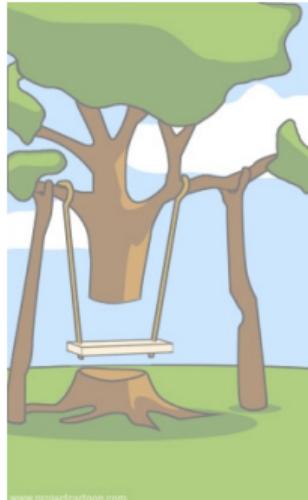
Design Patterns (Entwurfsmuster)



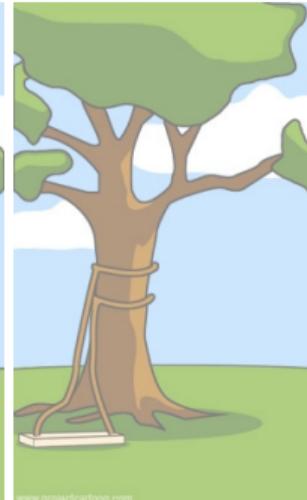
how the customer
explained it



how the project
leader understood it

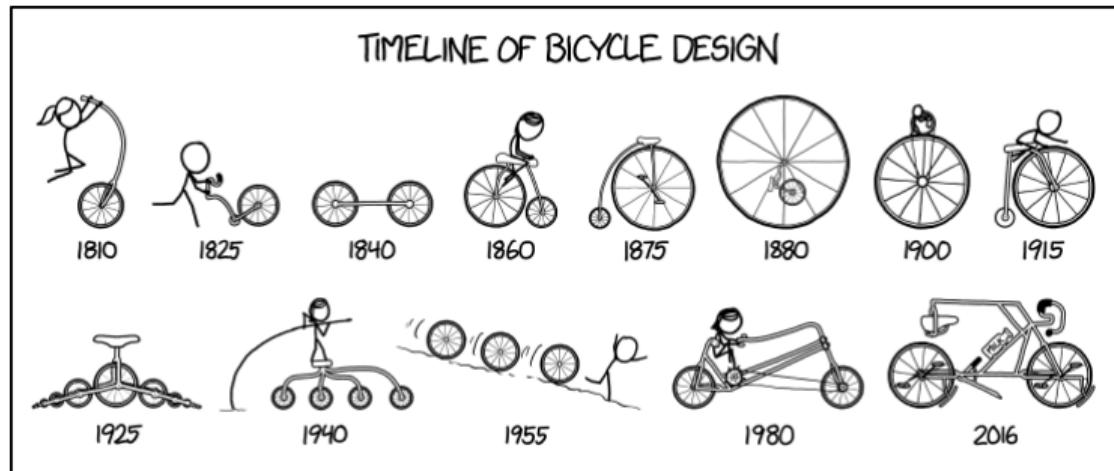


how the analyst
designed it



how the programmer
implemented it

Tired of Swings? What about Bikes?



Lecture Overview

1. Structural Patterns
2. Creational Patterns
3. Behavioral Patterns

Lecture Contents

1. Structural Patterns

Gang of Four

Design Patterns

Overview on Design Patterns

Object Adapter Pattern

Composite Pattern

Excursion: Show Complete Formula

Decorator Pattern

Example of the Decorator Pattern

Excursion: Introduce Necessary Brackets

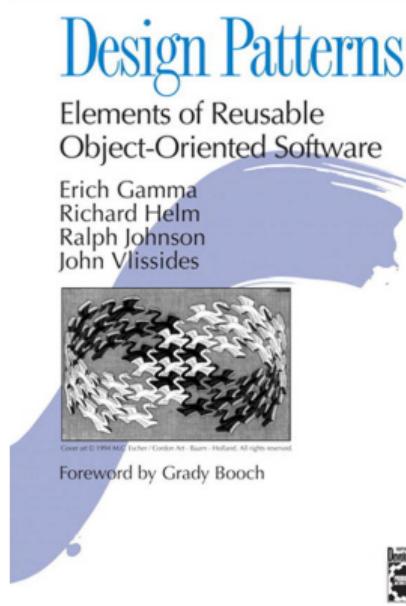
Overview on Design Patterns

Lessons Learned

2. Creational Patterns

3. Behavioral Patterns

Gang of Four [Gang of Four (GoF)]



Design Patterns

[Gang of Four]

Motivation

"Designing object-oriented software is hard, and designing **reusable** object-oriented software is even harder. [...] It takes a long time for novices to learn what good object-oriented design is all about. Experienced designers evidently know something inexperienced ones don't. What is it?"

Design Patterns (Entwurfsmuster)

pattern name for communication and high-level abstraction

problem when to apply the pattern

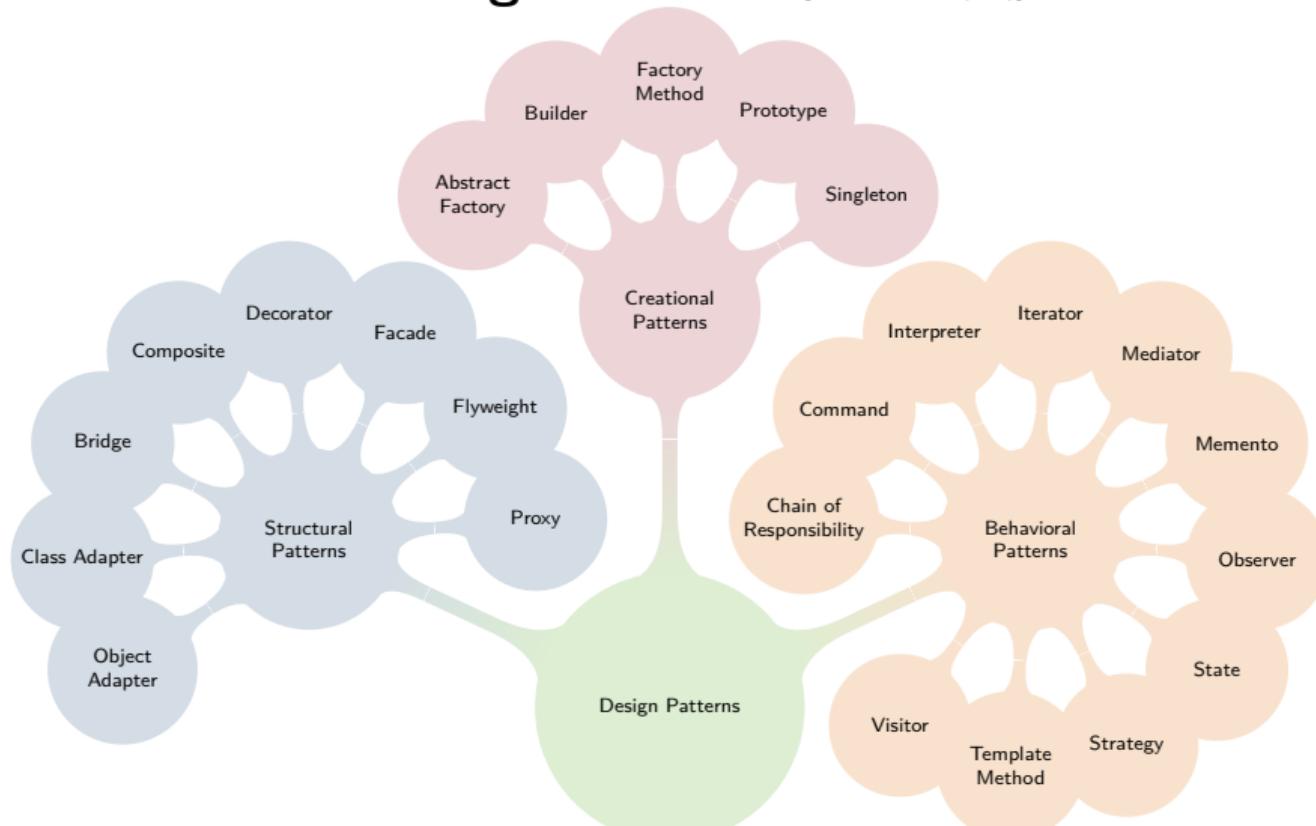
solution template on how to arrange classes and objects

consequences trade-offs of applying the pattern

Kinds of Patterns

"Creational patterns (**Erzeugungsmuster**) concern the process of object creation. Structural patterns (**Strukturmuster**) deal with the composition of classes or objects. Behavioral patterns (**Verhaltensmuster**) characterize the ways in which classes or objects interact and distribute responsibility."

Overview on Design Patterns [Gang of Four (GoF)]



Object Adapter Pattern

Object Adapter

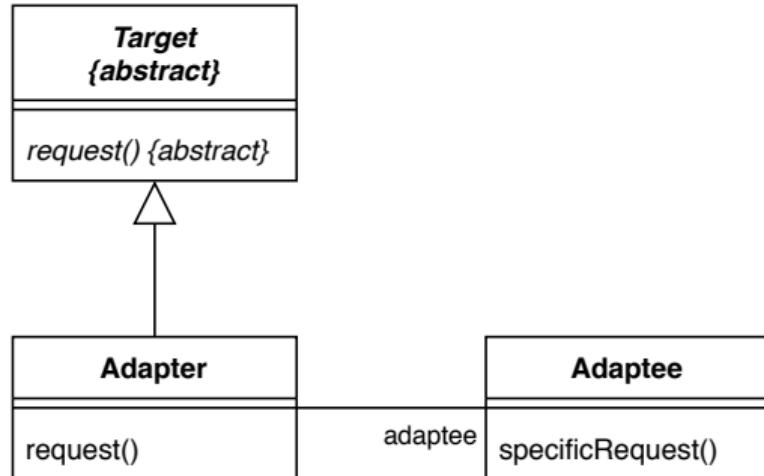
[Gang of Four]

intent “Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.”

aka. wrapper

motivation enable reuse of classes even though incompatible interfaces cannot be made compatible

idea create a new class with a compatible interface that forwards all requests



Composite Pattern

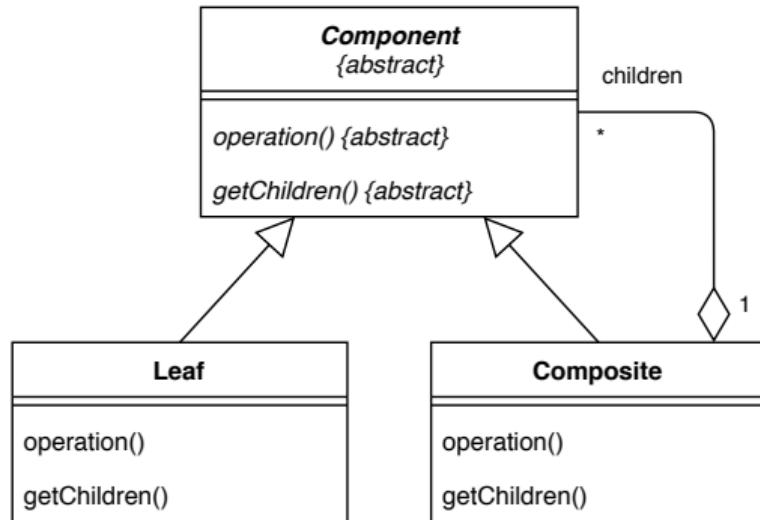
Composite

[Gang of Four]

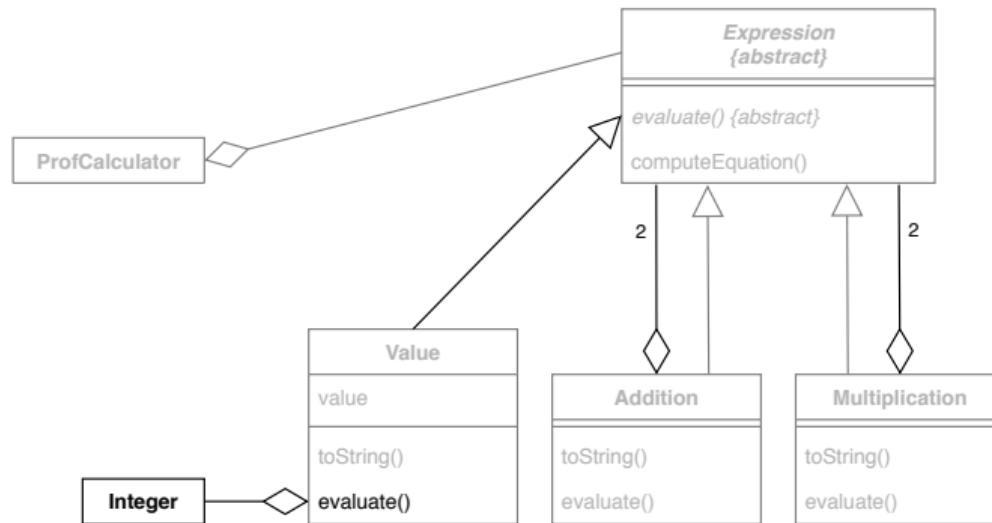
intent “Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.”

motivation avoid case distinctions for primitive and compound objects

idea create a common abstract super class enabling a unified access



Excursion: Show Complete Formula



Decorator Pattern

Decorator

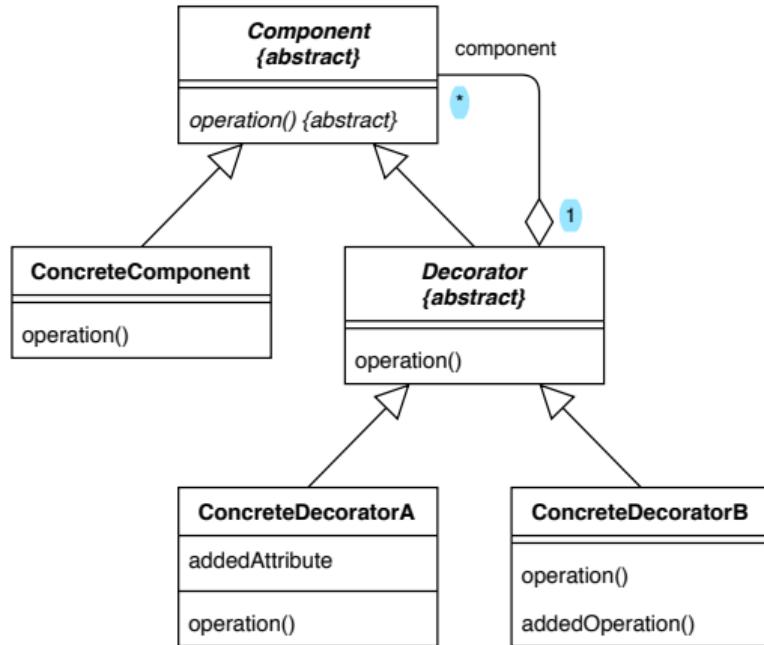
[Gang of Four]

intent “Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.”

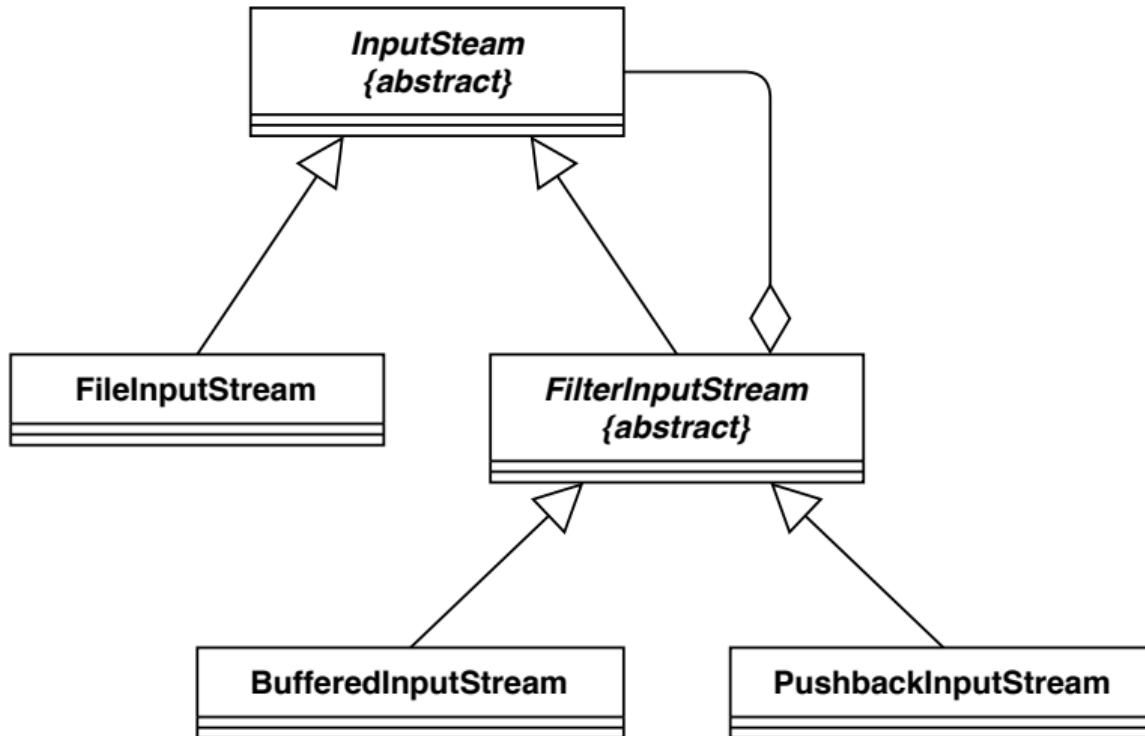
aka. wrapper (again)

motivation avoid explosion of static classes when combining all additional behaviors with all applicable classes

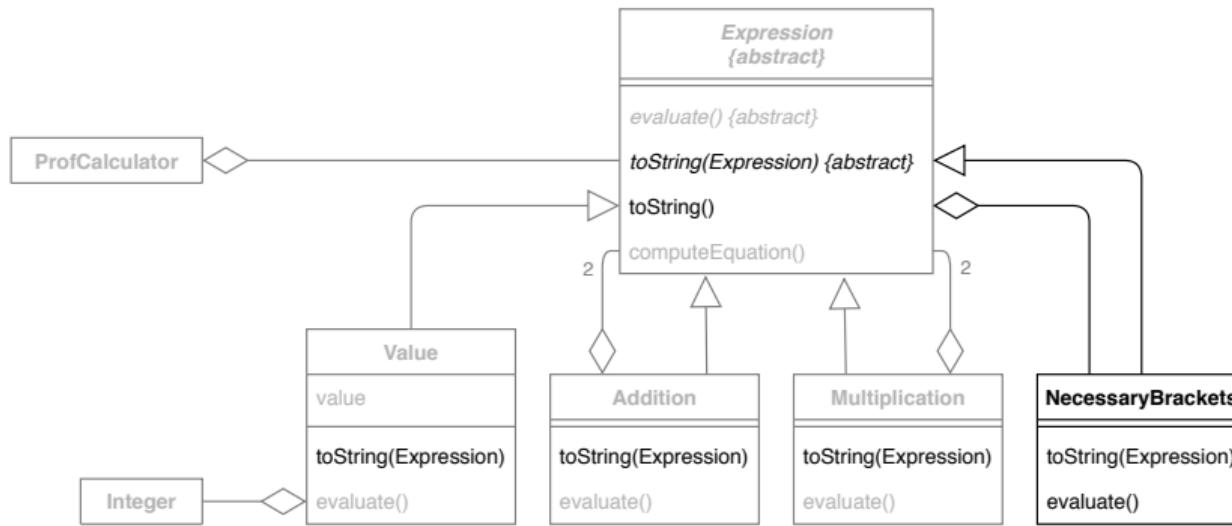
idea create decorators and components with the same interface, whereas decorators forward behavior whenever feasible



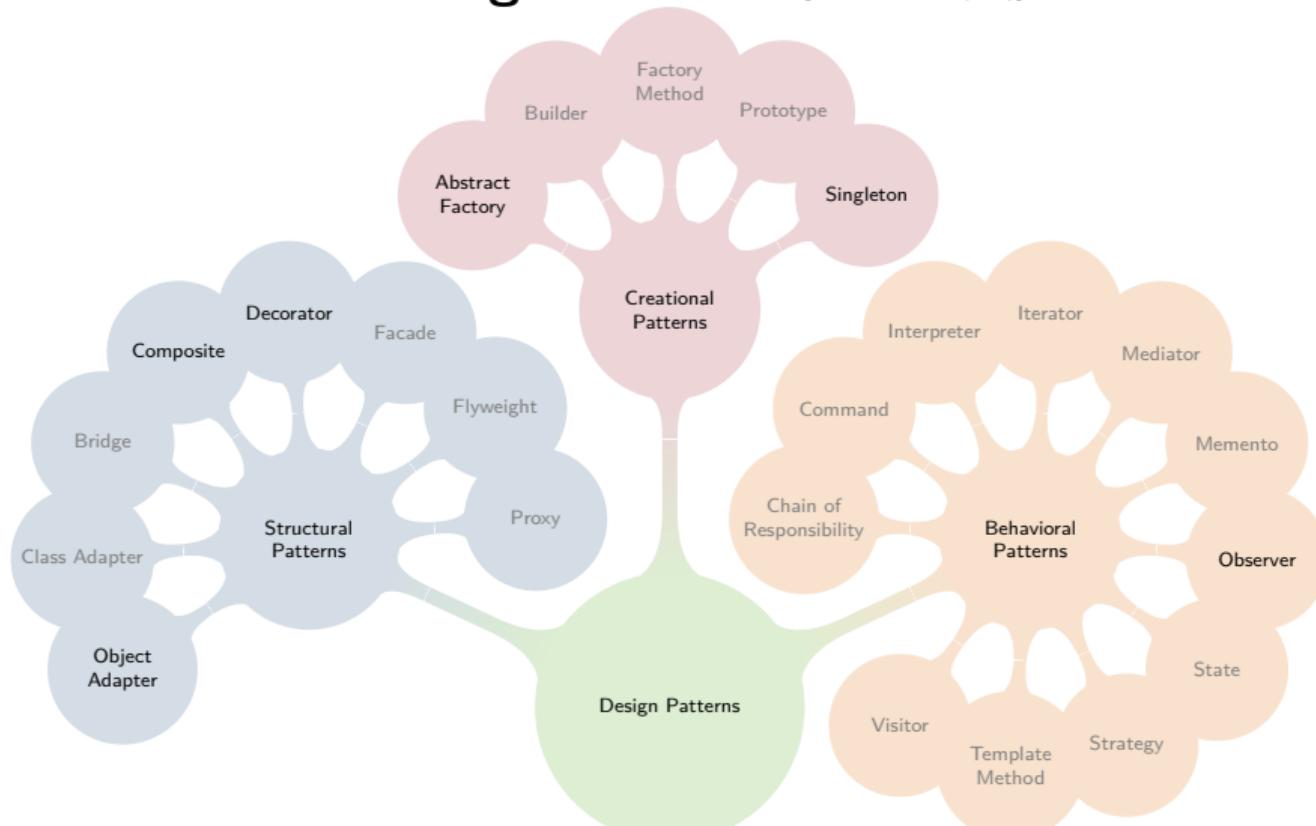
Example of the Decorator Pattern



Excursion: Introduce Necessary Brackets



Overview on Design Patterns [Gang of Four (GoF)]



Structural Patterns

Lessons Learned

- Design patterns by the Gang of Four
- Object adapter, composite pattern, decorator pattern
- Further Reading: [Gang of Four \(GoF\)](#), Chapter 4

Practice

- See [Moodle](#)
- Inspect code on Github: <https://github.com/tthuem/2020WS-SWT-Calculator/tree/xmaslecturev1>
- Implement further operations or the ability to show formulas without brackets and by using evaluation if needed (and checkbox to switch between brackets and evaluation)
- Fork the project on Github, upload your changes, and send pull request to get feedback

Lecture Contents

1. Structural Patterns

2. Creational Patterns

Singleton Pattern

Excursion: Add Logging for Debugging

Abstract Factory Pattern

Excursion: Create Styles for GUI

Overview on Design Patterns

Lessons Learned

3. Behavioral Patterns

Singleton Pattern

Singleton

[Gang of Four]

intent "Ensure a class [has only] one instance, and provide a global point of access to it."

motivation avoid the uncontrolled creation of multiple instances

idea a private constructor is called on class initialization and a public static method is used for access to that single instance

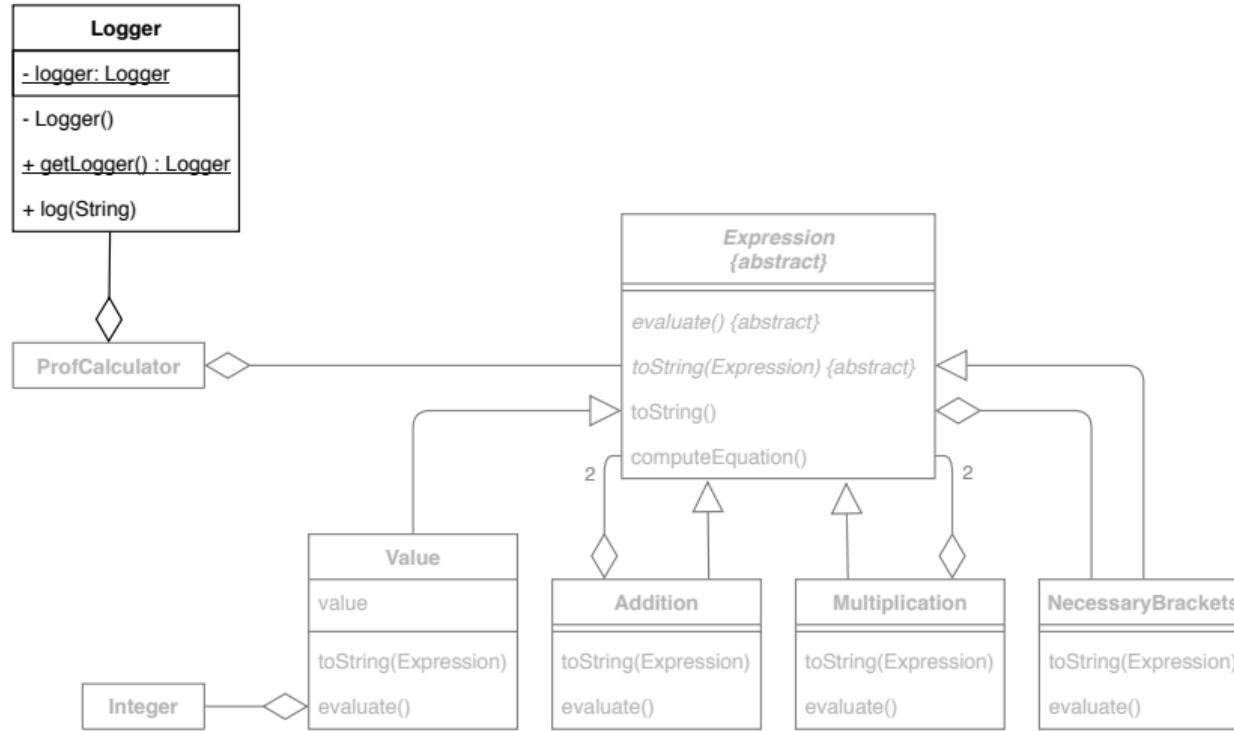
Singleton

- instance: Singleton

- Singleton()

+ getInstance(): Singleton

Excursion: Add Logging for Debugging



Abstract Factory Pattern

Abstract Factory

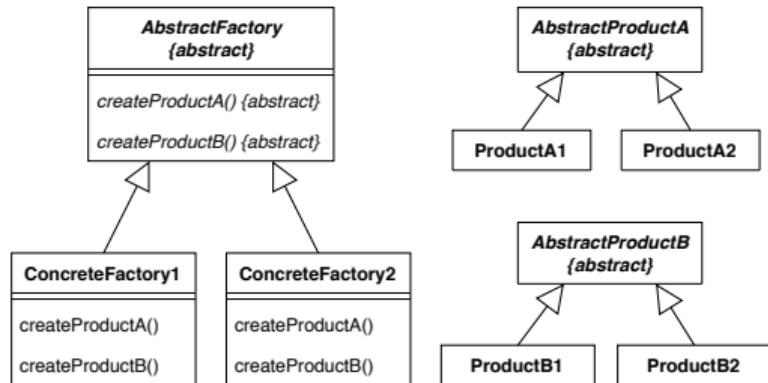
[Gang of Four]

intent “Provide an interface for creating families of related or dependent objects without specifying their concrete classes.”

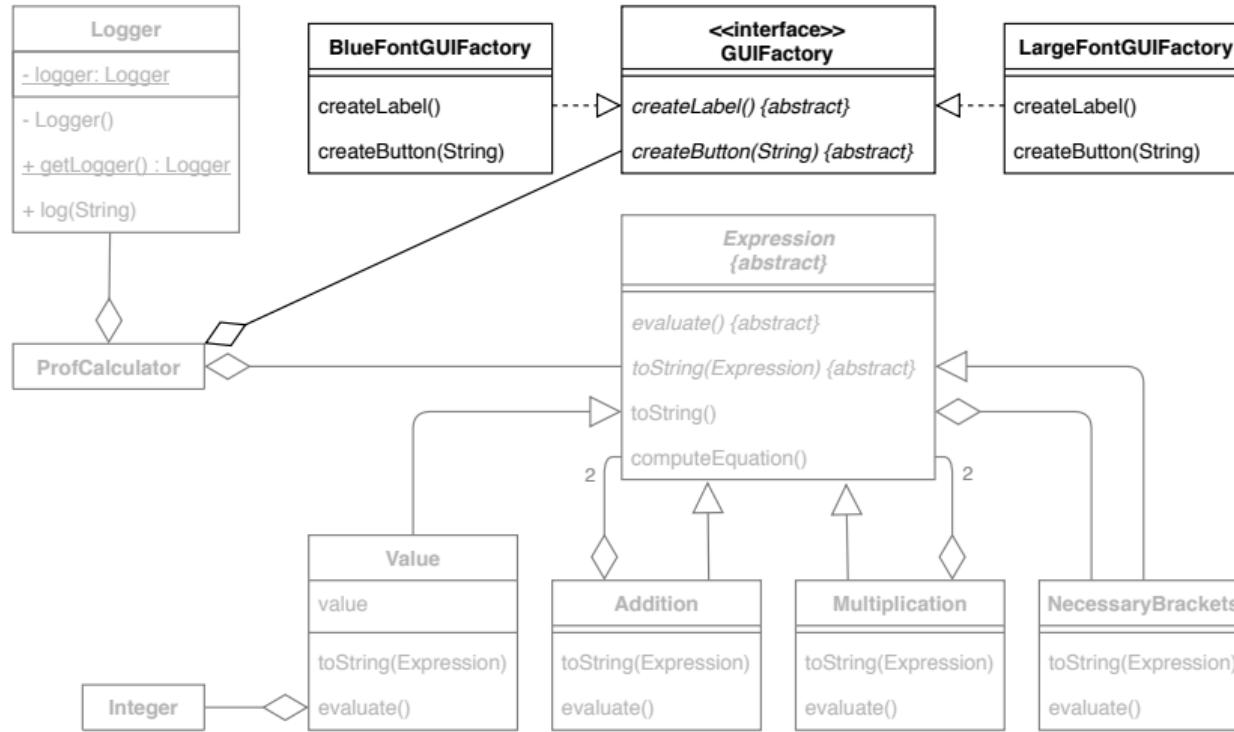
aka. kit

motivation avoid case distinctions when creating objects of certain kind, consistently create objects of a particular kind

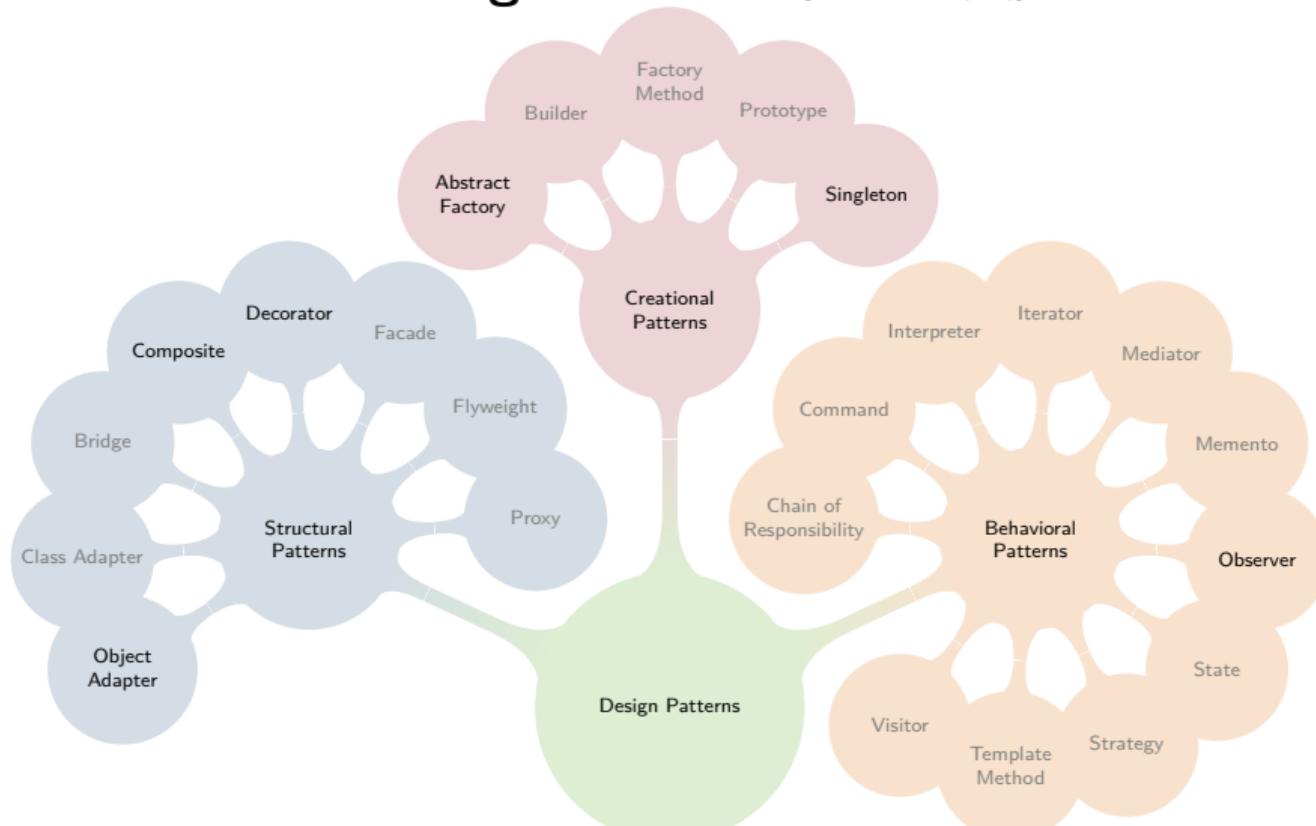
idea create classes for the consistent creation of objects



Excursion: Create Styles for GUI



Overview on Design Patterns [Gang of Four (GoF)]



Creational Patterns

Lessons Learned

- Singleton pattern, abstract factory pattern
- Further Reading: Gang of Four (GoF), Chapter 3

Practice

- See Moodle
- Inspect code on Github: <https://github.com/tthuem/2020WS-SWT-Calculator/tree/xmaslecturev2>
- Implement your own GUI factory or add the textbox to all factories
- Fork the project on Github, upload your changes, and send pull request to get feedback

Lecture Contents

1. Structural Patterns
2. Creational Patterns
3. Behavioral Patterns
 - Observer Pattern
 - Excursion: Counter from 1 to 10
 - Overview on Design Patterns
 - Lessons Learned

Observer Pattern

Observer

[Gang of Four]

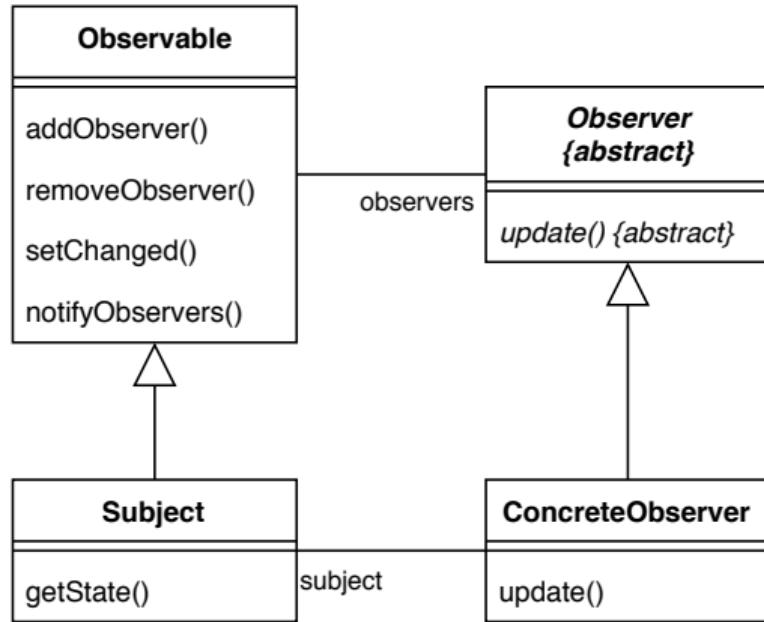
intent “Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.”

aka. dependents, publish-subscribe

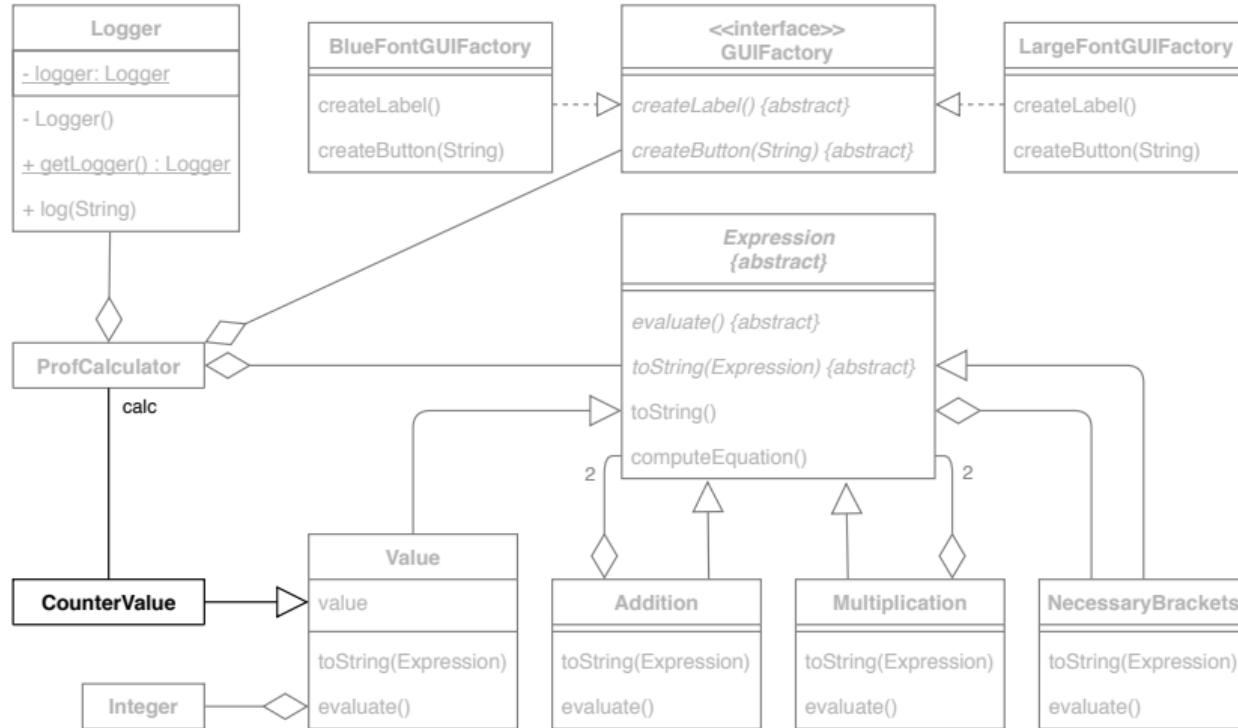
motivation avoid coupling of classes (i.e., explicit references such as imports)

example no coupling between model and view in model-view-controller architectures

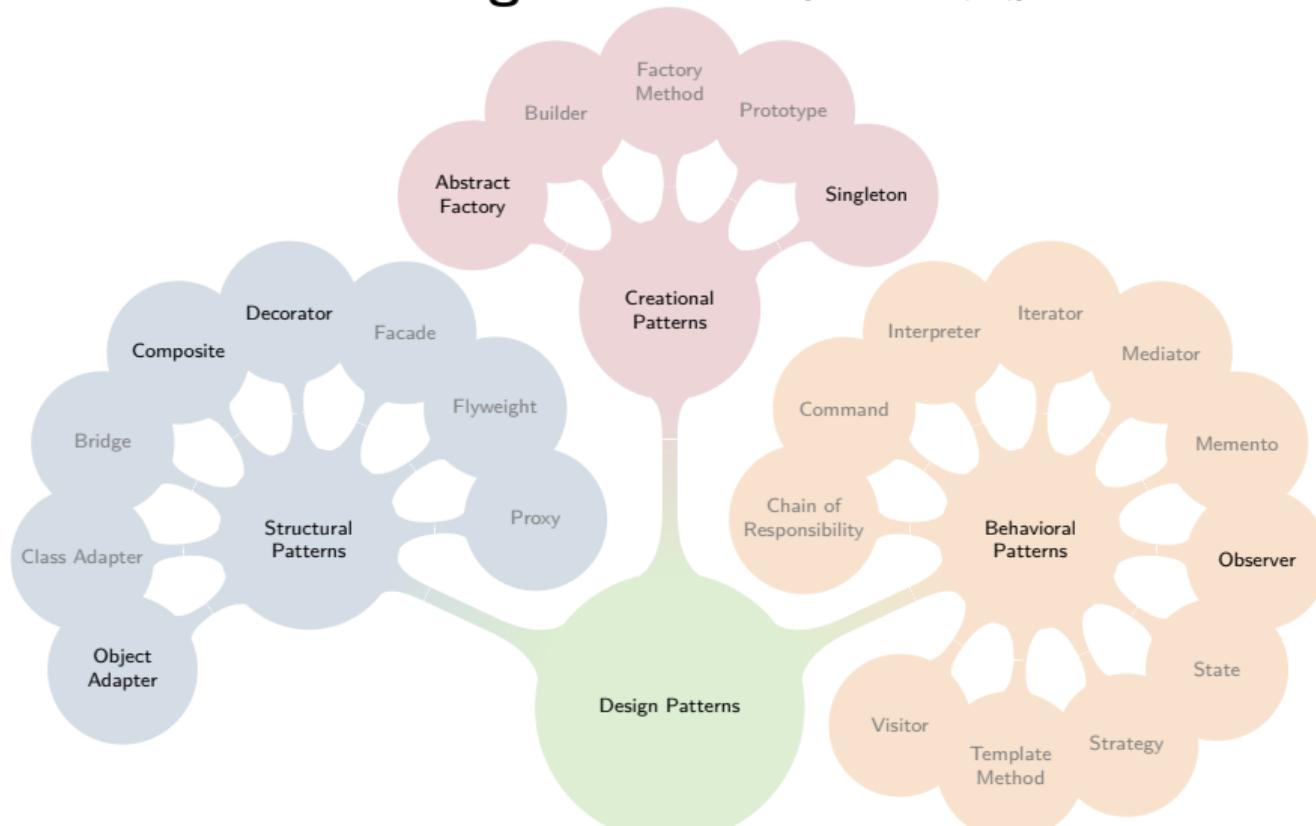
idea data object (model) has a list of observers (views) that are notified about changes



Excursion: Counter from 1 to 10



Overview on Design Patterns [Gang of Four (GoF)]



Behavioral Patterns

Lessons Learned

- Observer pattern
- Further Reading: Gang of Four (GoF), Chapter 5

Practice

- See Moodle
- Inspect code on Github: <https://github.com/tthuem/2020WS-SWT-Calculator/tree/xmaslecturev3>
- Implement the observer pattern (to avoid explicit reference of the calculator in CounterValue) or one other behavioral pattern not discussed in the lecture
- Fork the project on Github, upload your changes, and send pull request to get feedback