Programming in Java

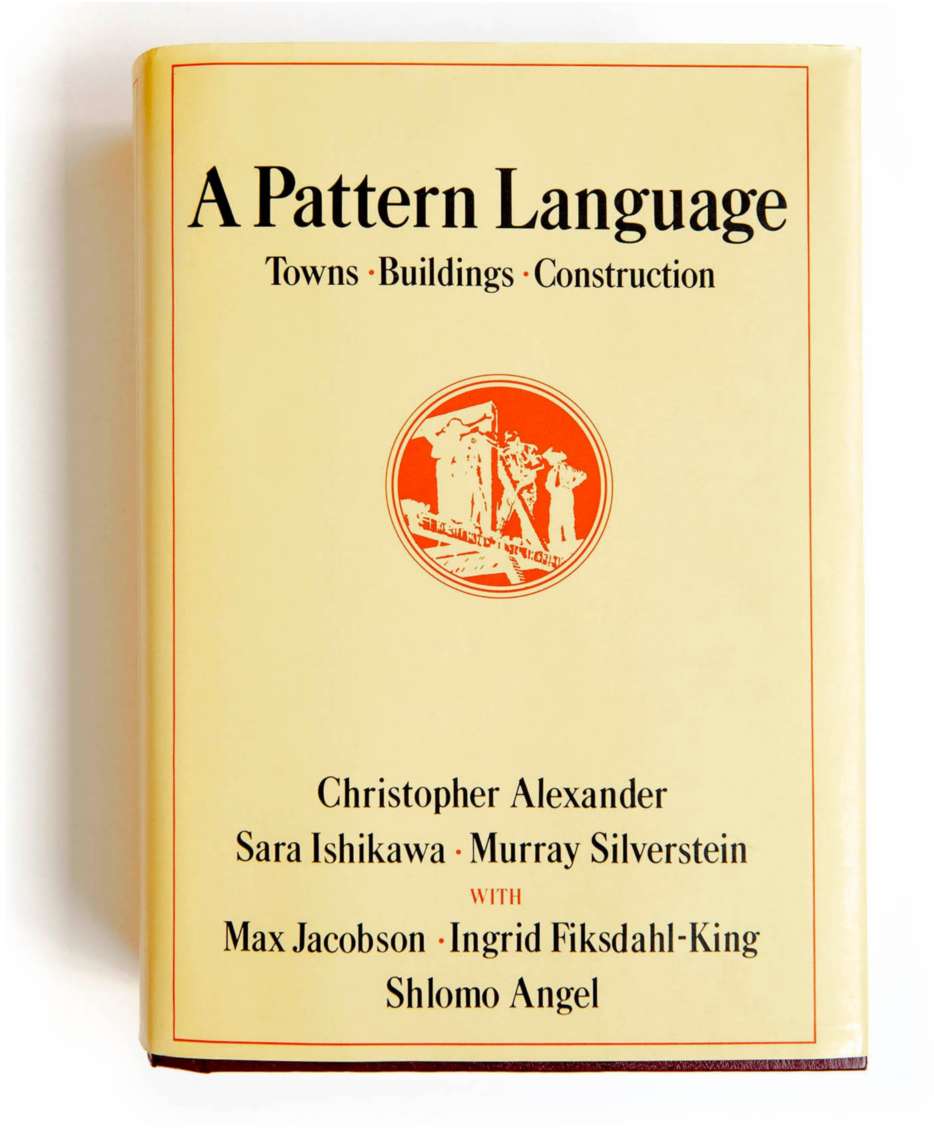
1. Design Patterns

Patterns

- The fundamental idea of patterns is that in any field
  - Effective solutions to similar types of problems tend to resemble each other
  - We can extract the common ideas and designs from the solutions as a generic pattern
  - Then we can apply this pattern to other similar problems by altering it to fit the problem
  - Not necessarily in the same domain
- For example
  - Solutions to engineering problems are often based on seeing how similar problems are solved by nature
  - Problem: How to handle complex machine learning tasks
  - Solution: Design a neural network that resembles how neurons work and are organized in the brain

Origins in Architecture

- The term "design pattern" originally comes from Christopher Alexander, an architect who in the 1970s published:
  - A Pattern Language: Towns, Buildings, Construction (1977)
  - Alexander described recurring solutions to design problems in architecture, each documented as a "pattern"
  - A reusable format that addresses context, problem, and solution.
  - These patterns encouraged human-centred, adaptable, and coherent design practices.

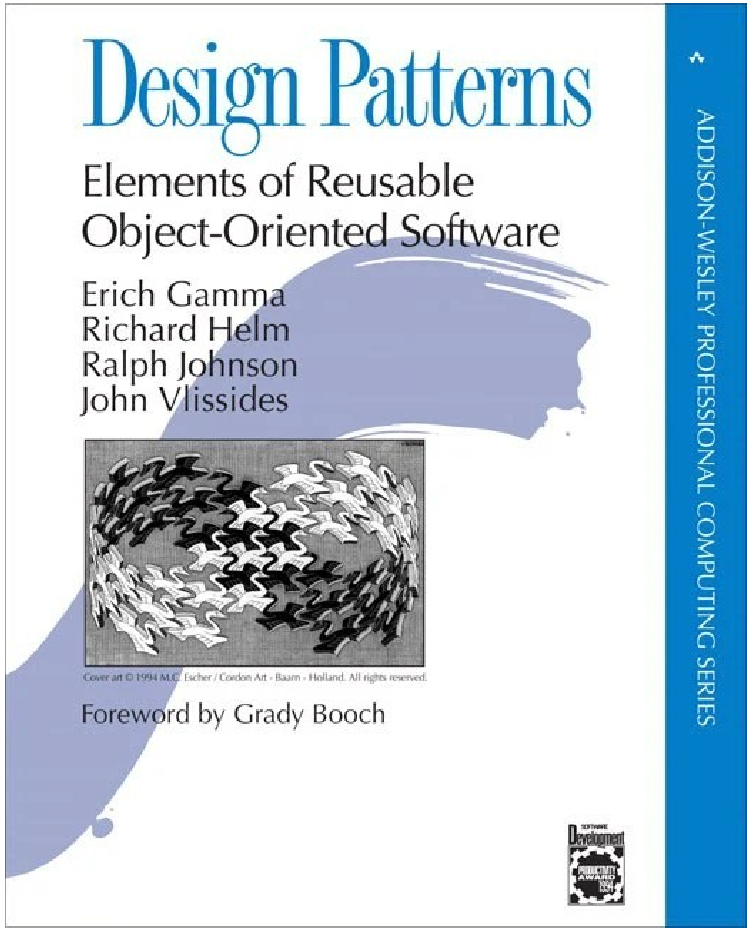

A Pattern Language
Towns · Buildings · Construction
Christopher Alexander
Sara Ishikawa · Murray Silverstein
WITH
Max Jacobson · Ingrid Fiksdahl-King
Shlomo Angel

OPSLA

- The OOPSLA (Object-Oriented Programming, Systems, Languages & Applications) conference, founded in the 1980s, became a hub for discussing emerging trends in object-oriented development.
  - At OOPSLA 1991, Kent Beck and Ward Cunningham presented "Using Pattern Languages for Object-Oriented Programs", explicitly linking Alexander’s ideas with software engineering.
  - OOPSLA served as the intellectual incubator for design patterns, Agile methods, and refactoring techniques.
- The Hillside Group, founded in 1993, further promoted pattern thinking through conferences like PLoP (Pattern Languages of Programs).

The Gang of Four (GoF)

- In the late 1980s and early 1990s, software engineers began applying Alexander’s ideas to software development.
- The most influential leap came with the 1994 book:
  - "Design Patterns: Elements of Reusable Object-Oriented Software" by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides — known as the Gang of Four (GoF).
  - Documented 23 classic patterns for object-oriented software design.
  - Categorized them into creational, structural, and behavioral patterns.
  - Emphasized flexibility, reuse, and decoupling.
  - Described in the context of C++ programming



Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

GofF Pattern Definition

- A GoF Design Pattern Consists of Four Essential Elements:
- Pattern Name
  - A short, memorable name that captures the essence of the pattern.
  - Enables developers to communicate design ideas concisely.
  - Example: Observer, Factory Method, Decorator.
- Problem
  - Describes when to apply the pattern
  - The context and conditions leading to the problem
  - And sometimes anti-patterns or forces that make it hard to resolve.
  - It clarifies the goal of the pattern and the design challenge it addresses.

GofF Pattern Definition

- Solution
  - Describes the elements (classes, objects, methods) that make up the design, their relationships, responsibilities, and collaborations.
  - It's not a complete implementation, but a template that can be adapted.
  - Includes static structure diagrams and sometimes pseudocode.
- Consequences
  - Discusses trade-offs, benefits, and potential drawbacks of using the pattern.
  - May cover issues like performance, flexibility, complexity, or code maintainability.
  - Helps assess the impact of the pattern on the system architecture.

Example: Observer Pattern

- Pattern Name: Observer
- Problem
  - How to ensure that when one object changes state, all its dependents are notified and updated automatically?
- Solution
  - Define a one-to-many dependency between objects.
  - The subject maintains a list of observers and notifies them of state changes.
- Consequences
  - Promotes loose coupling.
  - Easy to add new observers.
  - Can cause performance issues if many observers exist.

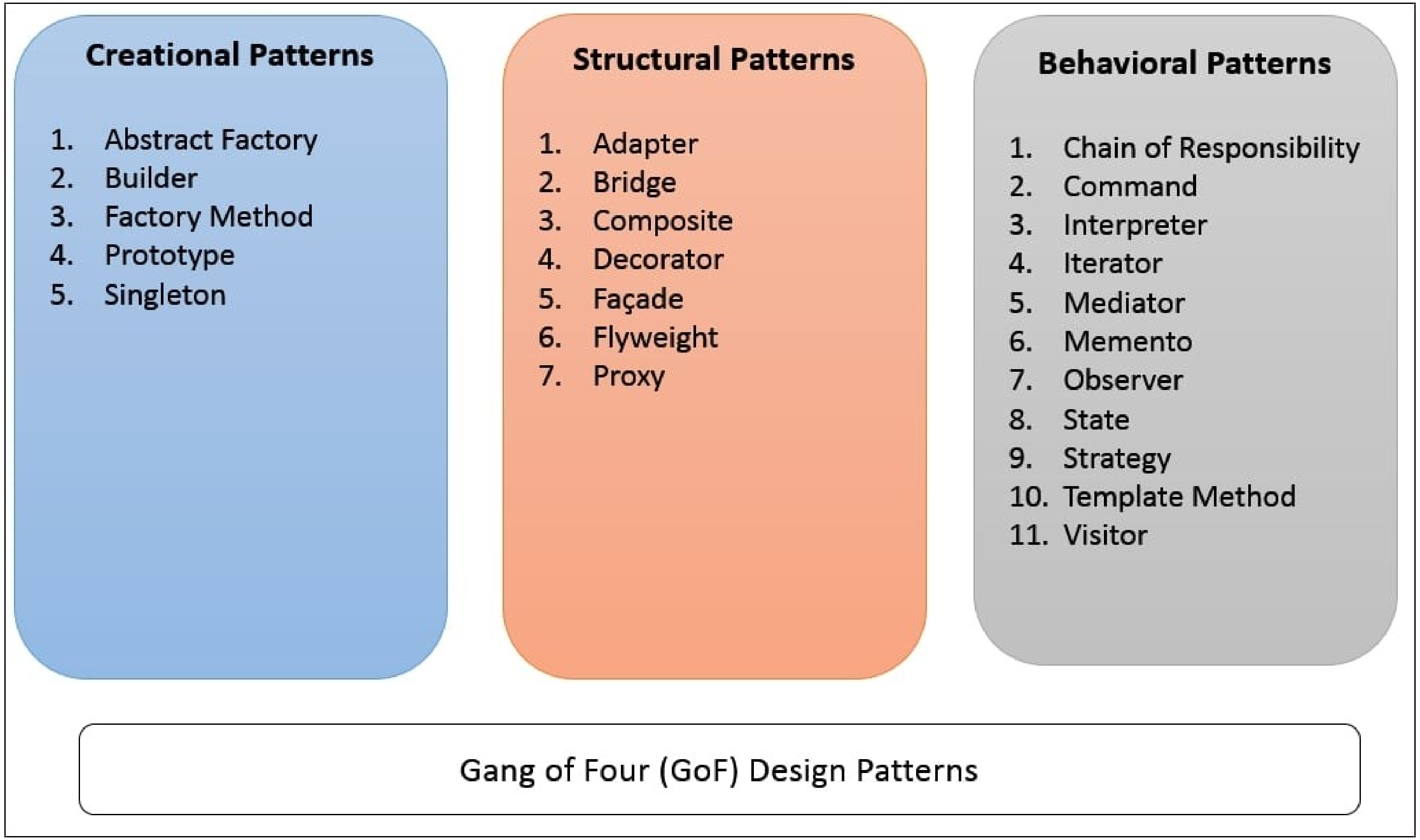
Example: Observer Pattern

- Pattern Name: Observer
- Problem
  - How to ensure that when one object changes state, all its dependents are notified and updated automatically?
- Solution
  - Define a one-to-many dependency between objects.
  - The subject maintains a list of observers and notifies them of state changes.
- Consequences
  - Promotes loose coupling.
  - Easy to add new observers.
  - Can cause performance issues if many observers exist.

GoF Design Pattern Categories and Their Focus

- Creational
  - - Concerned with object creation mechanisms.
  - Abstract the instantiation process, making it more flexible and decoupled.
  - Help manage the complexities of creating objects in a scalable and controlled way.
- Structural
  - Concerned with object and class composition.
  - Help ensure that if one part of a system changes, the entire structure doesn’t break.
  - Simplify relationships between entities and ensure good encapsulation.
- Behavioral
  - Concerned with object interaction and responsibility delegation.
  - Define how objects communicate and distribute responsibilities, promoting loose coupling and scalability in communication flows.

GoF Design Patterns



Creational Patterns

1. Abstract Factory
2. Builder
3. Factory Method
4. Prototype
5. Singleton

Structural Patterns

1. Adapter
2. Bridge
3. Composite
4. Decorator
5. Façade
6. Flyweight
7. Proxy

Behavioral Patterns

1. Chain of Responsibility
2. Command
3. Interpreter
4. Iterator
5. Mediator
6. Memento
7. Observer
8. State
9. Strategy
10. Template Method
11. Visitor

Gang of Four (GoF) Design Patterns

Creational Patterns

- Singleton
  - Ensures a class has only one instance and provides a global access point.
- Factory Method
  - Defines an interface for creating an object but lets subclasses decide which class to instantiate.
- Abstract Factory
  - Provides an interface for creating families of related or dependent objects without specifying concrete classes.
- Builder
  - Separates the construction of a complex object from its representation so the same construction process can create different representations.
- Prototype
  - Creates new objects by copying an existing object (a prototype), allowing for dynamic and flexible object creation.

Real World Creational Patterns

- Software pattern are often modelled after real world solutions that have nothing to do with software
- Singleton
  - The Pope. At any given time, there exists at most one object of type Pope
  - The person may change but there is only one persona of type Pope
- Factory Method
  - A Restaurant's Chef's special of the day. There exists a generic "chef" and a generic "special" that are abstract
  - A concrete chef "Maurice" will decide the concrete special to be created is "coq au vin"
  - While another concrete chef "Klaus" will decide the concrete special to be created is "Wiener schnitzel"

Real World Creational Patterns

- Builder
  - House to be built is specified by the blueprints provided by the architect
  - The specific steps needed to build the house are encapsulated in the General Contractor who is an object of type builder
  - For different types of houses, the builder object creates a custom series of steps to create it
- Prototype
  - A photo copier is uses the original document as a prototype
  - Then creates new documents from the prototype
- Abstract Factory
  - A vehicle assembly line that produces different vehicles: sports cars, SUVs and vans
  - All vehicles have a chassis, body, and engine, but the right combination has to be implemented for each type
  - You can’t put a SUV body on a sports car chassis

Creational Patterns in Java

- Prototype
  - The standard Object method `clone()` allows the creation of clones of objects when implemented in a class
  - This is a low level implementation of the Prototype pattern
- Builder
  - The StringBuilder class seen in the first week is an example of the Builder pattern
  - The way we compose streams to create a pipeline is also an example of the Builder pattern
- Factory Method
  - A Java interface defines an abstract method
  - When implemented in a class, each class provides a different concrete implementation

Lab 1-1

Creational Patterns

Structural Patterns

- Adapter
  - Converts one interface into another that clients expect, allowing incompatible interfaces to work together.
- Bridge
  - Decouples an abstraction from its implementation so they can vary independently.
- Composite
  - Composes objects into tree structures to represent part-whole hierarchies.
  - Clients can treat individual objects and compositions uniformly.
- Decorator
  - Adds responsibilities to objects dynamically without altering their structure.

Structural Patterns

- Facade
  - Provides a simplified interface to a complex subsystem.
- Flyweight
  - Reduces memory usage by sharing as much data as possible with similar objects.
- Proxy
  - Provides a placeholder or surrogate for another object to control access to it.

Structural Patterns Real World Examples

- Adapter
  - A common example of an adapter in the plug adapter that change converts a US type appliance plug to a a European style plug
- Bridge
  - Examples of physical interfaces are a light switch and a dimmer switch
  - Examples of implementations are a overhead light and a ceiling fan
  - We can use mix and match interfaces and implementation, we can use a light switch with a ceiling fan and a dimmer switch with a light
- Composite
  - One of the most common example of this is a file system where
  - Directories contain files
  - But directories are also files, so directories can contain directories
  - Or “we can put things in boxes, including other boxes”

Structural Patterns Real World Examples

- Decorator
  - This is a very common pattern in the Java I/O class library
  - A terrain map is an object
  - A clear plastic sleeve is a decorator for map
  - Wrapping the map in the plastic adds the functionality that we can now write on the map and then erase the writing later
  - We have wrapped a map in a decorator to produce an erasable mapW

Structural Patterns Real World Examples

- Facade
  - A good example of a facade is the receptionist for a company
  - The receptionist determines who you need to see then contacts that person
  - You don’t need to know anything about the company, just how to interact with the receptionist
- Flyweight
  - This is used when we have a limited number of expensive objects but we know that only a fraction of the users will be using those objects at the same time
  - A common application is where we have a motor pool of cars and drivers that can be used by executives, and we know that at any given time, only 10% of the executives will need a driver.
  - If we have 100 executives, we create a pool of 15 cars and drivers
  - When an executive needs a driver, they check one out from the motor pool and return it to the pool when they are done with it

Structural Patterns Real World Examples

- Proxy
  - A proxy is a stand in.
  - If you are waiting in line to buy tickets to a show, but need to leave for a moment
  - You pay someone $10 to keep your place in line as a proxy for you.
  - If they get to the front of the line, they are told to text you so you can come back and take your place

Lab 1-2

Structural Patterns

Behavioral Patterns

- Chain of Responsibility
  - Passes a request along a chain of handlers, where each handler can choose to process the request or pass it on.
  - We have seen an example of this with exceptions.
  - If a try{} block doesn’t catch an exception, the exception passes to any higher level ecnlosing try{} block
- Command
  - Encapsulates a request as an object, allowing parameterization of clients and queuing or logging of requests.
  - Essentially turns a function call into an object
- Interpreter
  - Defines a grammar for a language and provides an interpreter to deal with its syntax.

Behavioral Patterns

- Iterator
  - Provides a way to access elements of a collection sequentially without exposing its underlying representation.
  - We have seen an implementation this in Java Collections
- Mediator
  - Defines an object that encapsulates how a set of objects interact, promoting loose coupling.
  - Replaces the need for objects to spend resources establishing peer connections
- Memento
  - Captures and restores an object’s internal state without violating encapsulation.
  - We have seen an implementation of this when we serialized an object

Behavioral Patterns

- Observer
  - Defines a one-to-many dependency between objects so that when one object changes state, all dependants are notified.
- State
  - Allows an object to alter its behavior when its internal state changes
  - The object appears to change its class.
- Strategy
  - Defines a family of algorithms, encapsulates each one, and makes them interchangeable.
- Template Method
  - Defines the skeleton of an algorithm, deferring some steps to subclasses.
  - Usually used with the Strategy pattern

Behavioral Patterns

- Visitor
  - Represents an operation to be performed on elements of an object structure, allowing new operations without changing the classes.
  - Essentially takes a common piece of functionality, like writing to an XML file, and puts the different variants for different classes in a server type object
  - Avoids needing to change existing classes
  - Each class just asks the visitor to perform the action for them
  - Useful when the functionality is volatile, then updates are required in only one place

Behavioral Patterns Real World Examples

- Chain of Responsibility
  - Consider a ticket opened on a help desk.
  - The initial person is not allowed to respond, so they escalate it to the next level
  - This will keep happening until someone (hopefully) handles it
  - If not implemented correctly, tickets can get stuck at some point of the chain
  - Or fall off the en
- Command
  - When taking a an order from a customer, a server in a restaurant, they will use the command pattern by writing it down
  - The written order object can then be passed to the chef or other staff asychronously for handling
  - Often used to support the Chain of Responsibility

Behavioral Patterns Real World Examples

- Interpreter
  - Musial score notation is an example of an interpreter.
  - It provides a notation and a grammar to encode and decode music
- Iterator
  - An iterator object encapsulates the logic for finding the next item in a collection
  - A triage nurse in an emergency room is an iterator
  - They apply a set of rules to determine who gets to see the doctor next
  - Different iterator logic than “first come, first in to see the doctor”

Behavioral Patterns Real World Examples

- Mediator
  - In a courtroom, all communications have to through the judge who acts as a mediator
  - Allowing people to talk directly to each other without restraint results in total chaos
  - Similarly, an airport control tower is a mediator that relays and coordinates communications with aircraft
- Memento
  - A backup of a file system is a Memento pattern
  - At a given point, a snapshot of the file system is taken as a memento
  - At any point in the future, the file system can be reverted to any one of the backup snapshots

Behavioral Patterns Real World Examples

- Observer
  - A subscriber mailing list is an example this pattern
  - The observers are the subscribers
  - The subject is the entity sending out the mails
  - If any observer or other event causes a change in state in the subject
  - The subject updates every observer
- State
  - A car shows the State pattern in determining what happens when the accelerator is pressed
  - If the car is in an off state, nothing happens
  - If the car is running and in gear, the car speeds up
  - If the car is running and not in gear, the engines revs

Behavioral Patterns Real World Examples

- Strategy
  - In a kitchen, the different ways of cooking meat – roasting, frying, etc – are strategies
  - Each may be recorded in a “how do” document
  - The chef can choose which one to use based on what sort of meat needs to be cooked
- Template Method
  - When applying for job, there might be a standard application process (the Template)
  - However depending on the positions being applied for, different types of interviews might be required (the slots in the template)
  - Note that they types of interviews also represent a Strategy pattern

Behavioral Patterns Real World Examples

- Visitor
  - Also referred to as the “consultant” pattern
  - A tax accountant is a visitor object
  - They known how to do the taxes for different types of people
  - Eliminates the need for people to also have to know the tax law that applies to them
  - They just ask the tax consultant to process their taxes for them

Other Patterns

- The idea of patterns has spread from OO programming to many other types of software engineering
  - Has become a standard way of documenting best practices and reusable solution to recurring problems
- For example: Real Time Systems
  - Active Object Pattern: Manages concurrency while hiding thread management from clients.
  - Scheduler Pattern: Prioritizing tasks under timing constraints
  - Watchdog Pattern: Ensure system reliability by monitoring whether critical components or tasks are responsive and behaving correctly within a time limit.
- Idioms
  - Design patterns that describe best practices unique to a specific programming language are referred to as “idioms”
  - The Pimpl idiom, which stands for “Pointer to Implementation,” is a C++ design technique used to hide the implementation details of a class from its users.
  - This would not apply to Java because Java does not use pointers

Lab 1-3

Behavioral Patterns

Java™