Behavioral Design Patterns

Software Modeling Foundations

Author: Eng. Carlos Andrés Sierra, M.Sc.
cavirguezs@udistrital.edu.co

Computer Engineer
Lecturer
Universidad Distrital Francisco José de Caldas

2024-III

Outline

Outline

Basic Concepts of Behavioral Patterns

- **Intent**: Focus on how classes distribute responsibilities among them, and at the same time each class just does a single cohesive function. It is like a *F1 Pits Team*, each one has a **single responsability**, but all together creates a complete team workflow.
- **Motivation**:
  - **Problem**: A system should be configured with multiple algorithms, and a system should be independent of how its operations are performed.
  - **Solution**: Define each algorithm, encapsulate each one, and make them work together.

Basic Concepts of Behavioral Patterns

- **Intent**: Focus on how classes distribute responsibilities among them, and at the same time each class just does a single cohesive function. It is like a *F1 Pits Team*, each one has a **single responsability**, but all together creates a complete team workflow.
- **Motivation**:
  - **Problem**: A system should be configured with multiple algorithms, and a system should be independent of how its operations are performed.
  - **Solution**: Define each algorithm, encapsulate each one, and make them work together.

Basic Concepts of Behavioral Patterns

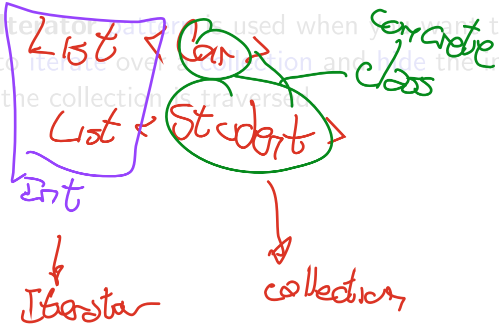
- **Intent**: Focus on how classes distribute responsibilities among them, and at the same time each class just does a single cohesive function. It is like a *F1 Pits Team*, each one has a **single responsability**, but all together creates a complete team workflow.
- **Motivation**:
  - **Problem**: A system should be configured with multiple algorithms, and a system should be independent of how its operations are performed.
  - **Solution**: Define each algorithm, encapsulate each one, and make them work together.

Outline

Outline

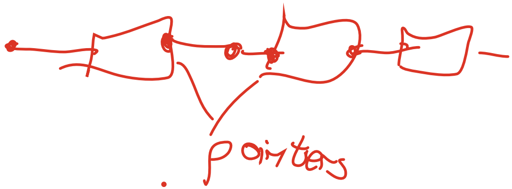
Iterator Pattern — Concepts

- The **Iterator** pattern is a behavioral pattern that allows sequential access to the elements of an aggregate object without exposing its underlying representation.
- The **Iterator** pattern is used when you want to provide a standard way to iterate over a collection and hide the implementation details of *how* the collection is traversed.



Iterator Pattern — Concepts

- The **Iterator** pattern is a behavioral pattern that allows sequential access to the elements of an aggregate object without exposing its underlying representation.
- The **Iterator** pattern is used when you want to provide a standard way to iterate over a collection and hide the implementation details of *how* the collection is traversed.



Iterator Pattern — Classes Structure

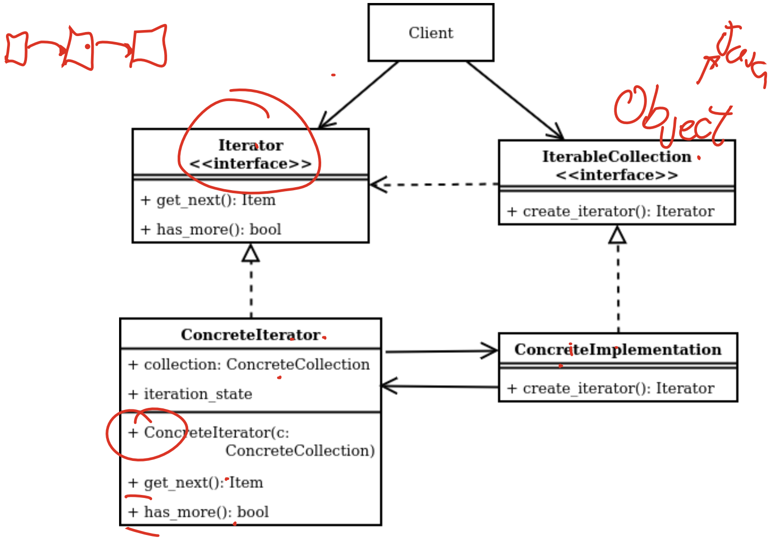
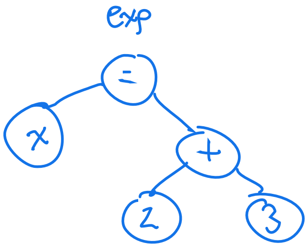


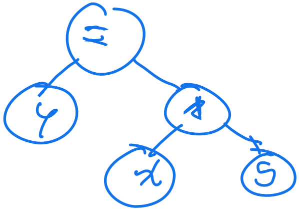
Figure: Iterator Pattern Class Diagram

Iterator Pattern Example: A Syntax Tree

$\underbrace{x = 2 + 3}$

$y = x * 5$





Outline

Memento Pattern — Concepts

- The **Memento** pattern is a behavioral pattern that lets you save and restore the previous state of an object without revealing the details of its implementation.
- The Memento pattern is used when you want to provide the ability to restore an object to its previous state (undo).
- The Memento pattern is used when you want to provide a rollback mechanism in case of errors or exceptions.

undo

- Google Docs
- Git

Memento Pattern — Concepts

- The **Memento** pattern is a behavioral pattern that lets you save and restore the previous state of an object without revealing the details of its implementation.
- The **Memento** pattern is used when you want to provide the ability to restore an object to its previous state (`undo`).
- The **Memento** pattern is used when you want to provide a rollback mechanism in case of errors or exceptions.

Z

[illegible]

Memento Pattern — Concepts

- The **Memento** pattern is a behavioral pattern that lets you save and restore the previous state of an object without revealing the details of its implementation.
- The **Memento** pattern is used when you want to provide the ability to restore an object to its previous state (`undo`).
- The **Memento** pattern is used when you want to provide a rollback mechanism in case of errors or exceptions.

Memento Pattern — Classes Structure

Sometimes going back to the past is the **best** way to fix the **future**.

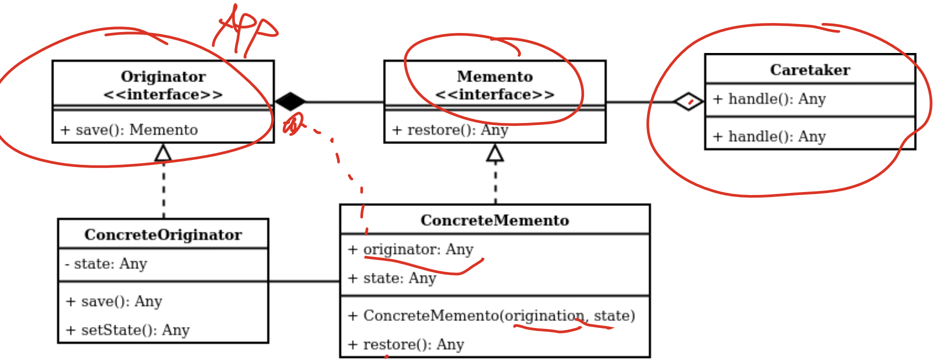
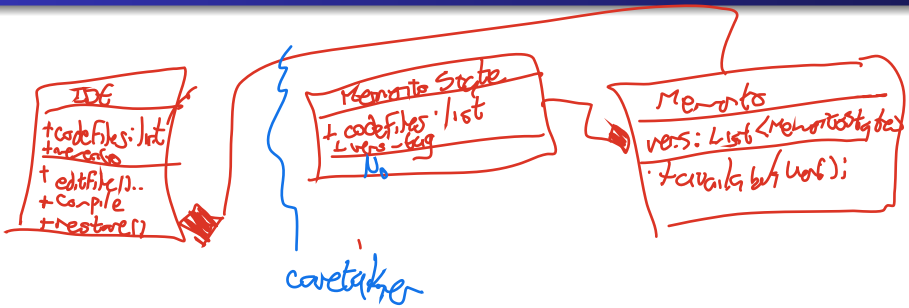


Figure: Memento Pattern Class Diagram

Memento Pattern Example: Versioning Your Code



Outline

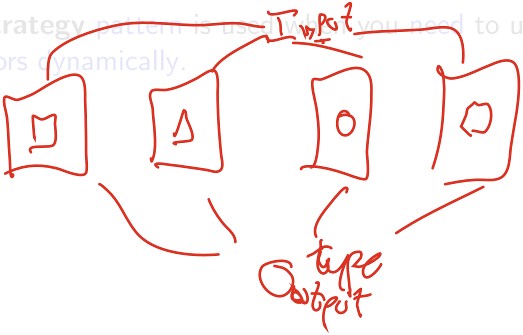
UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

Strategy Pattern — Concepts

- The **Strategy** pattern is a behavioral pattern that lets yo define a family of algorithms, put each of them into a separate class, and make their $\boxed{\text{objects interchangeable}}$.
- The **Strategy** pattern is used when you want to define a class that will have one behavior that is similar to other behaviors in a list.
- The **Strategy** pattern is used when you need to use **one** of several behaviors dynamically.

Strategy Pattern — Concepts

- The **Strategy** pattern is a behavioral pattern that lets yo define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.
- The **Strategy** pattern is used when you want to define a class that will have one behavior that is similar to other behaviors in a list.
- The **Strategy** pattern is used when you need to use **one** of several behaviors dynamically.



Strategy Pattern — Concepts

- The **Strategy** pattern is a behavioral pattern that lets yo define a family of algorithms, put each of them into a separate class, and make their objects interchangeable.
- The **Strategy** pattern is used when you want to define a class that will have one behavior that is similar to other behaviors in a list.
- The **Strategy** pattern is used when you need to use **one** of several behaviors dynamically.

Strategy Pattern — Classes Structure

In a set of similar problems, you can choose the **best strategy** to solve it.

request {
if cond1: return cs1.handle;
else if cond2: cs2.handle;

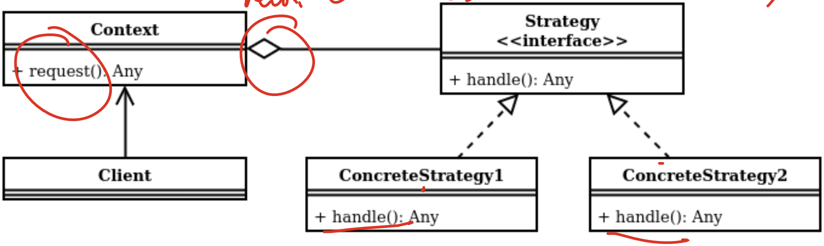
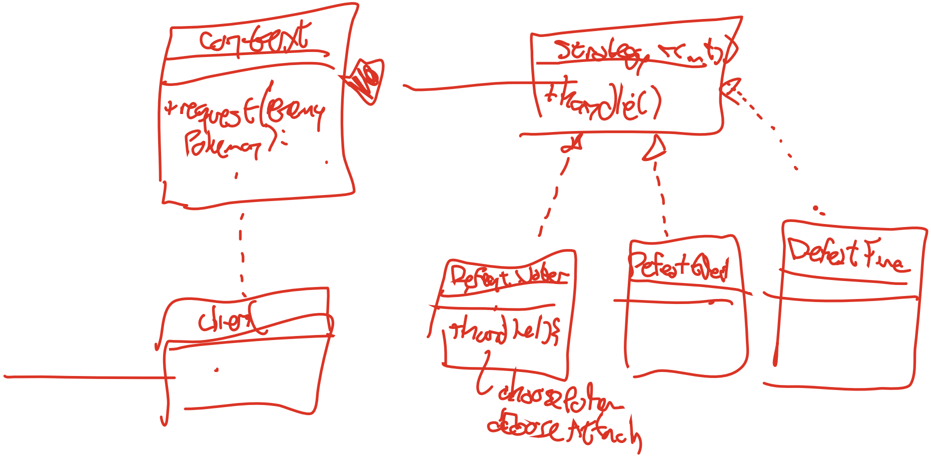


Figure: Strategy Pattern Class Diagram

Strategy Pattern Example: Be a Pokemon Trainer



Outline

Template Pattern — Concepts

- The **Template** pattern is a behavioral pattern that defines the program skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure.
- The **Template** pattern is used when you want to let clients extend only specific steps of an algorithm, but **not the whole algorithm** or its structure.
- The **Template** pattern is used when you have several classes that contain the same set of methods, but you want to avoid code duplication.

Template Pattern — Concepts

- The **Template** pattern is a behavioral pattern that defines the program skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure.
- The **Template** pattern is used when you want to let clients extend only specific steps of an algorithm, but **not the whole algorithm** or its structure.
- The **Template** pattern is used when you have several classes that contain the same set of methods, but you want to avoid code duplication.

Template Pattern — Concepts

- The **Template** pattern is a behavioral pattern that defines the program skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure.
- The **Template** pattern is used when you want to let clients extend only specific steps of an algorithm, but **not the whole algorithm** or its structure.
- The **Template** pattern is used when you have several classes that contain the same set of methods, but you want to avoid code duplication.

Template Pattern — Classes Structure

Somethings are always the same, but some things are always different.

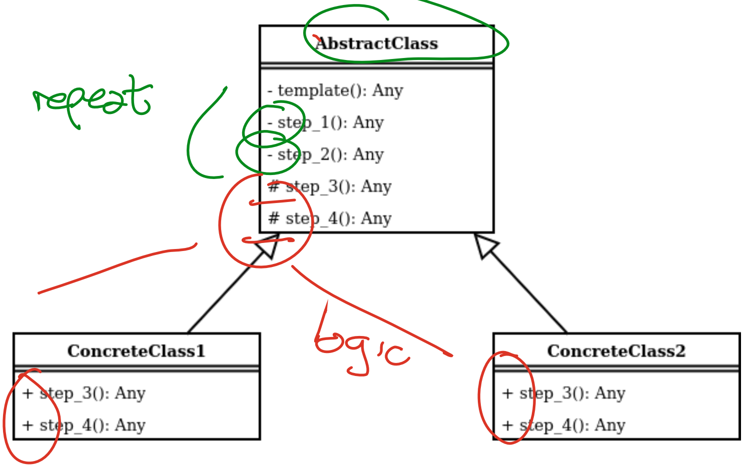
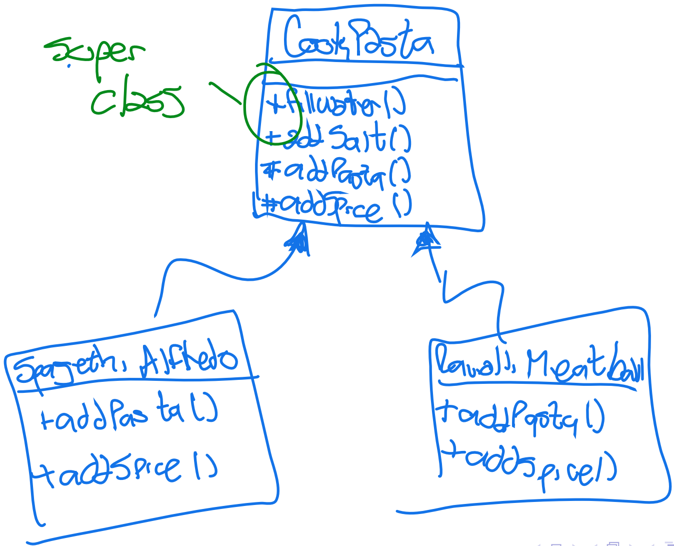


Figure: Template Pattern Class Diagram

Template Pattern Example: Let's Cook Pasta!

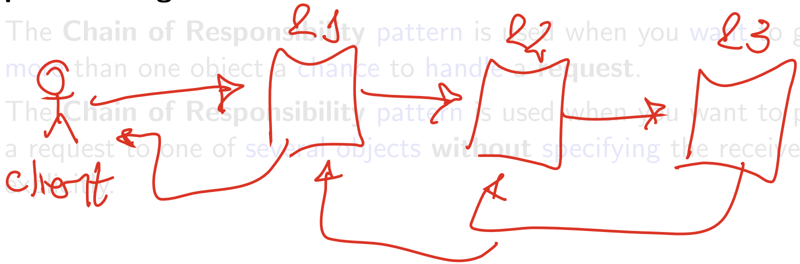


Outline

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

Chain of Responsability Pattern — Concepts

- The **Chain of Responsibility** pattern is a behavioral pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to **process the request** or **to pass it along the chain**.
- The **Chain of Responsibility** pattern is used when you want to give more than one object a chance to handle a request.
- The **Chain of Responsibility** pattern is used when you want to pass a request to one of several objects **without** specifying the receiver explicitly.



Chain of Responsability Pattern — Concepts

- The **Chain of Responsibility** pattern is a behavioral pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to **process the request** or **to pass it along the chain**.
- The **Chain of Responsibility** pattern is used when you want to give more than one object a chance to handle a **request**.
- The **Chain of Responsibility** pattern is used when you want to pass a request to one of several objects **without** specifying the receiver explicitly.

Chain of Responsability Pattern — Concepts

- The **Chain of Responsibility** pattern is a behavioral pattern that lets you pass requests along a chain of handlers. Upon receiving a request, each handler decides either to **process the request** or **to pass it along the chain**.
- The **Chain of Responsibility** pattern is used when you want to give more than one object a chance to handle a **request**.
- The **Chain of Responsibility** pattern is used when you want to pass a request to one of several objects **without** specifying the receiver explicitly.

Chain of Responsability Pattern — Classes Structure

A lot of quality reviewers are needed to approve a **high quality product**.

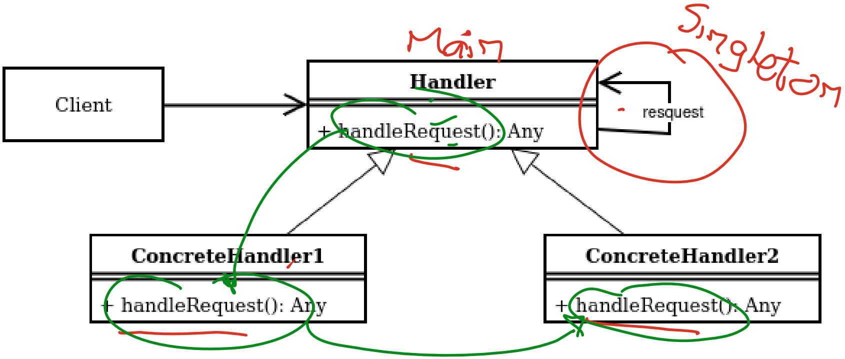
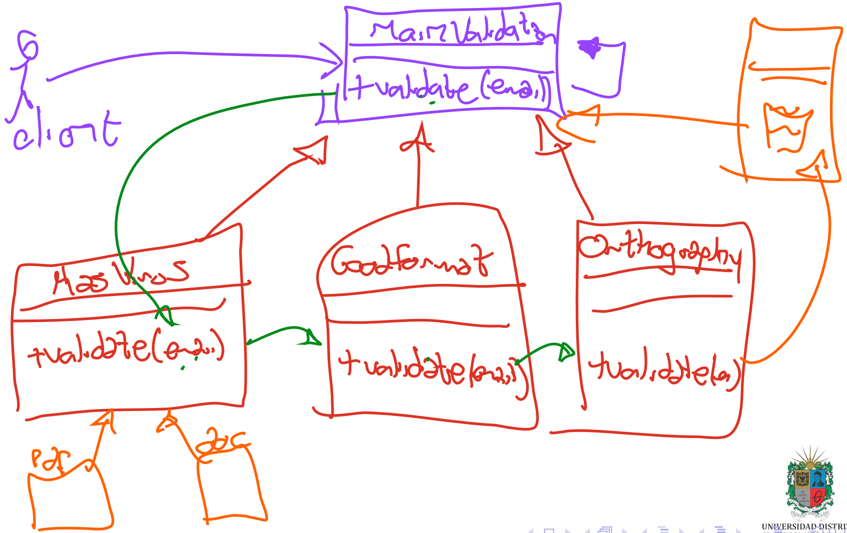


Figure: Chain of Responsability Pattern Class Diagram

Chain of Responsability Pattern Example: Filter an Email



Outline

State Pattern — Concepts

attributes → state
behaviors

- The **State** pattern is a behavioral pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.
- The **State** pattern is used when you want to have an object that behaves as if it were an instance of a different class when its internal state changes.
- The **State** pattern is used when you want to avoid a large number of conditional statements in your code.

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

State Pattern — Concepts

- The **State** pattern is a behavioral pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.
- The **State** pattern is used when you want to have an object that behaves as if it were an instance of a different class when its internal state changes.
- The **State** pattern is used when you want to avoid a large number of conditional statements in your code.

State Pattern — Concepts

- The **State** pattern is a behavioral pattern that lets an object alter its behavior when its internal state changes. It appears as if the object changed its class.
- The **State** pattern is used when you want to have an object that behaves as if it were an instance of a different class when its internal state changes.
- The **State** pattern is used when you want to avoid a large number of conditional statements in your code.

State Pattern — Classes Structure

You **never** act the **same** when you are happy or sad.

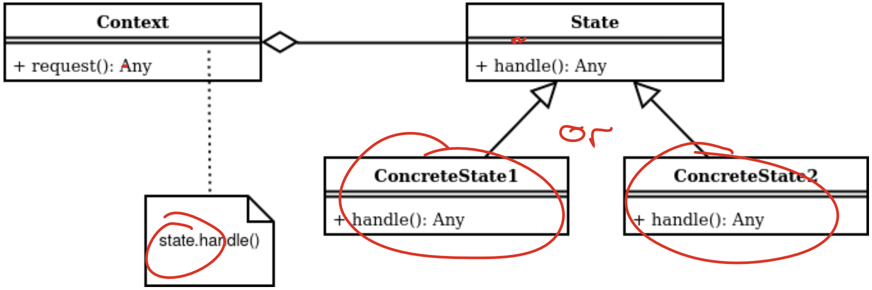
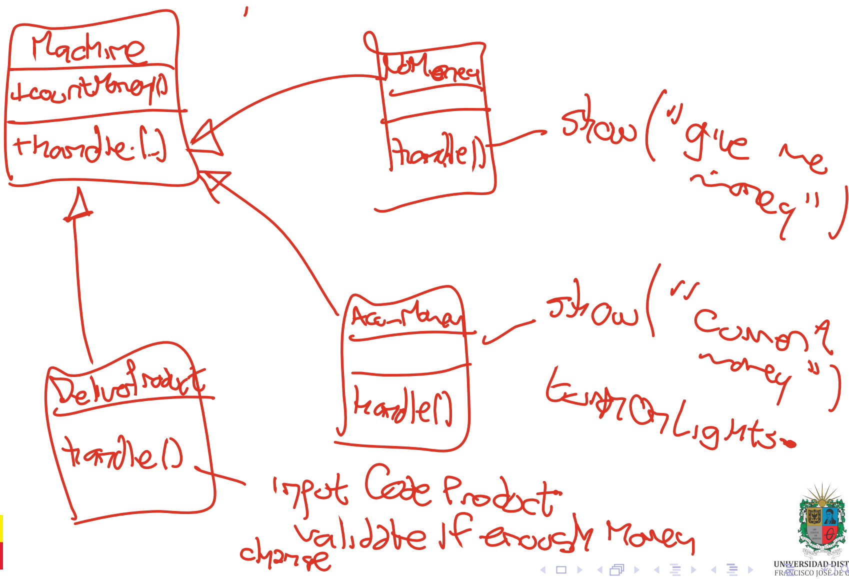


Figure: State Pattern Class Diagram

State Pattern Example: Vending Machine



Outline

Mediator Pattern — Concepts

Hexagonal Architectures - DDD

- The **Mediator** pattern is a behavioral pattern that lets you reduce chaotic dependencies between objects. The pattern restricts direct communications between the objects and forces them to collaborate only via a mediator object.
- The **Mediator** pattern is used when you want to reduce the number of dependencies between your classes.
- The **Mediator** pattern is used when you want to simplify the communication between objects in a system.

Mediator Pattern — Concepts

- The **Mediator** pattern is a behavioral pattern that lets you reduce chaotic dependencies between objects. The pattern restricts direct communications between the objects and forces them to collaborate only via a mediator object.
- The **Mediator** pattern is used when you want to reduce the number of dependencies between your classes.
- The **Mediator** pattern is used when you want to simplify the communication between objects in a system.

Mediator Pattern — Concepts

- The **Mediator** pattern is a behavioral pattern that lets you reduce chaotic dependencies between objects. The pattern restricts direct communications between the objects and forces them to collaborate only via a mediator object.
- The **Mediator** pattern is used when you want to reduce the number of dependencies between your classes.
- The **Mediator** pattern is used when you want to simplify the communication between objects in a system.

Mediator Pattern — Classes Structure

Maybe you just need to call a **mediator** to solve your problems.

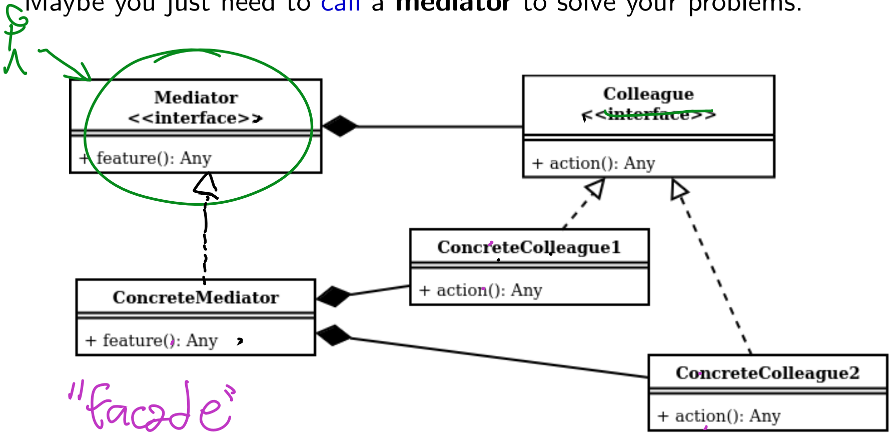
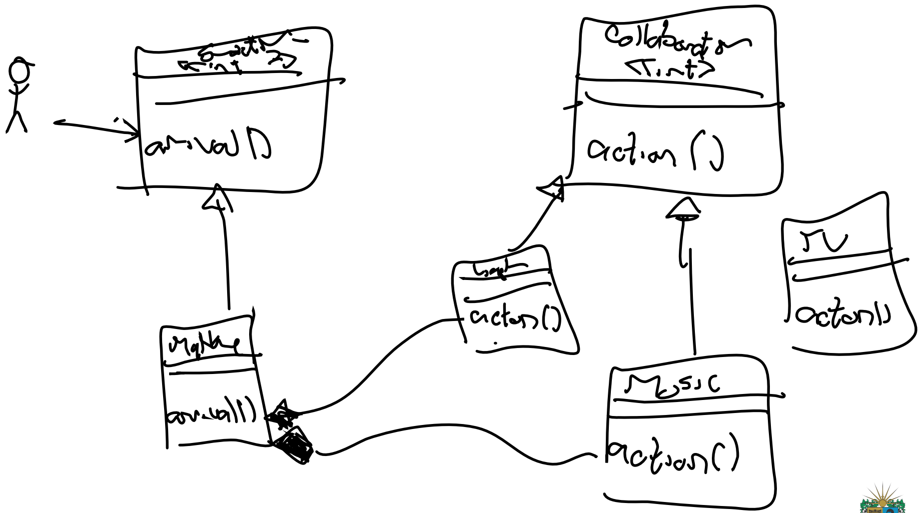


Figure: Mediator Pattern Class Diagram

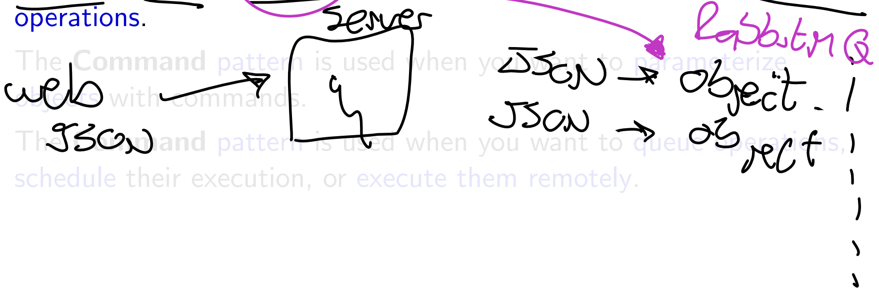
Mediator Pattern Example: Smart Home



Outline

Command Pattern — Concepts

- The **Command** pattern is a behavioral pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method argument, delay or queue a request's execution, and support undoable operations.
- The **Command** pattern is used when you want to parameterize objects with commands.
- The **Command** pattern is used when you want to queue operations, schedule their execution, or execute them remotely.



Command Pattern — Concepts

- The **Command** pattern is a behavioral pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method argument, delay or queue a request's execution, and support undoable operations.
- The **Command** pattern is used when you want to parameterize objects with commands.
- The **Command** pattern is used when you want to queue operations, schedule their execution, or execute them remotely.

Command Pattern — Concepts

- The **Command** pattern is a behavioral pattern that turns a request into a stand-alone object that contains all information about the request. This transformation lets you pass requests as a method argument, delay or queue a request's execution, and support undoable operations.
- The **Command** pattern is used when you want to parameterize objects with commands.
- The **Command** pattern is used when you want to queue operations, schedule their execution, or execute them remotely.

cron jobs

* * * * *

day min hou day
week

Command Pattern — Classes Structure

Since the beginning of time, commands have been given to people to be executed.

Declarative - what? no how

Select * From: Students;

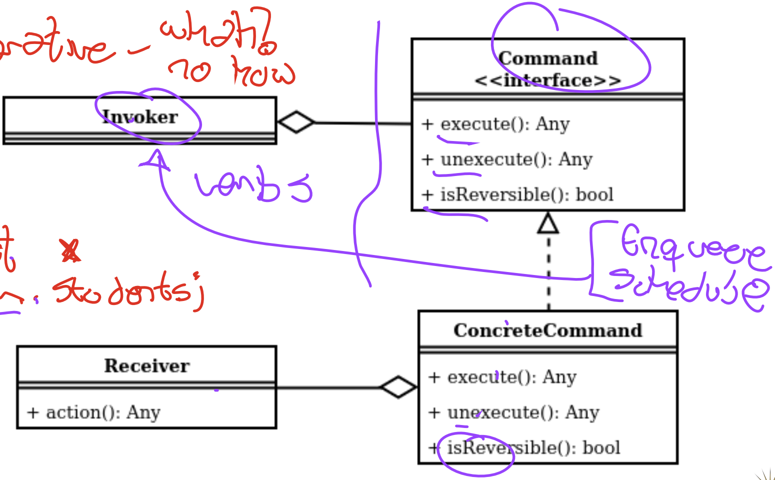
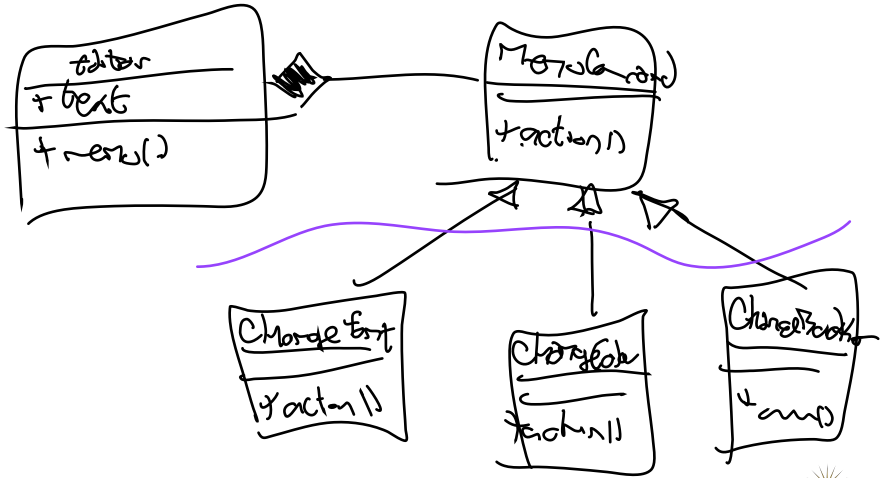


Figure: Command Pattern Class Diagram

Command Pattern Example: Your Own Text Editor

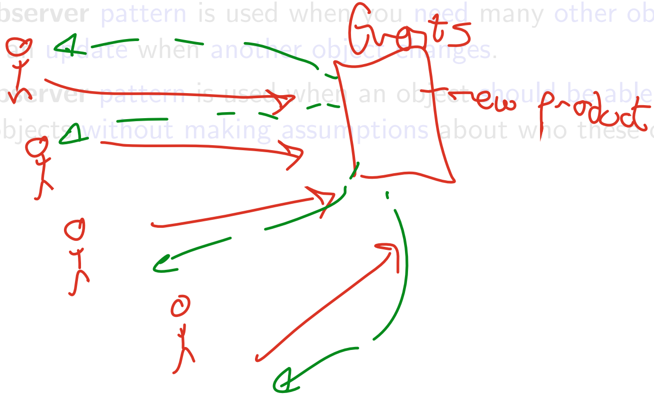


Outline

1. Introduction
2. Patterns
   - Iterator 9
   - Memento 8
   - Strategy* 3
   - Template 7
   - Chain of Responsability* 2 ✓ ✓
   - State 6
   - Mediator 5
   - Command* 4
   - Observer* 1 ✓
3. Conclusions

Observer Pattern — Concepts

- The **Observer** pattern is a behavioral pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.
- The **Observer** pattern is used when you need many other objects to receive an update when another object changes.
- The **Observer** pattern is used when an object should be able to notify other objects without making assumptions about who these objects are.



Observer Pattern — Concepts

- The **Observer** pattern is a behavioral pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.
- The **Observer** pattern is used when you need many other objects to receive an update when another object changes.
- The **Observer** pattern is used when an object should be able to notify other objects without making assumptions about who these objects are.

Observer Pattern — Concepts

- The **Observer** pattern is a behavioral pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.
- The **Observer** pattern is used when you need many other objects to receive an update when another object changes.
- The **Observer** pattern is used when an object should be able to notify other objects without making assumptions about who these objects are.

Observer Pattern — Classes Structure

When you have a lot of eyes looking at you, you are an **observer**.

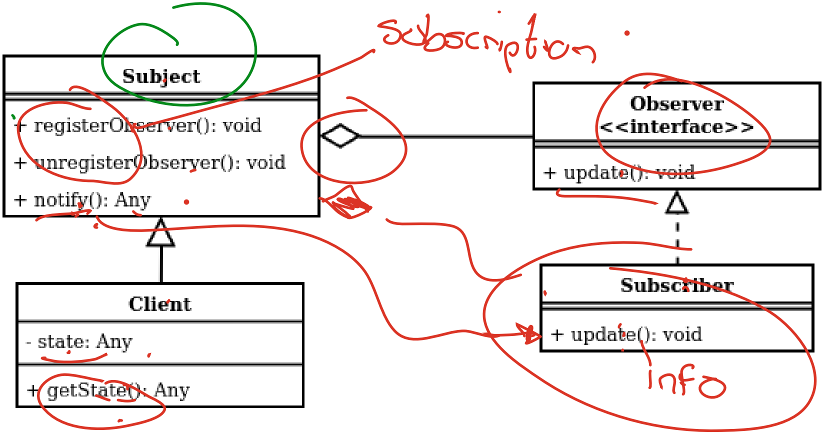
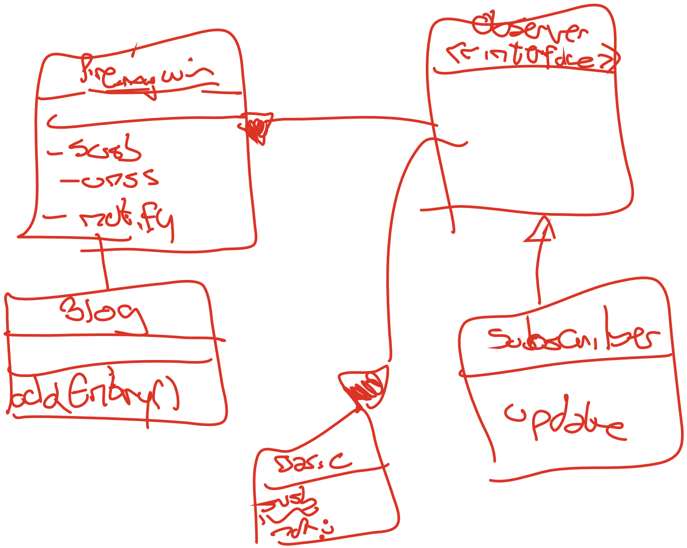


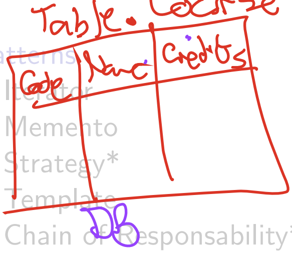
Figure: Observer Pattern Class Diagram

Observer Pattern Example: Blogs!



Outline

DAO → Data Access Object



```
class. Course() {   Ent.
    int code;
    str name;
    int credits;
}
```

ORMs

```
class CourseDAO() {
    def save();
    def get();
    def delete();
}
```

Conclusions

- **Behavioral Patterns** are a set of patterns that focus on how objects distribute responsibilities among them.
- **Behavioral Patterns** are used when you want to provide a standard way to iterate over a collection, save and restore the previous state of an object, define a family of algorithms, alter an object's behavior when its internal state changes, reduce chaotic dependencies between objects, turn a request into a stand-alone object, define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.
- **Behavioral Patterns** are not recommended when you have a system that doesn't change, or you have a system that doesn't have a lot of objects.

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS

Conclusions

- **Behavioral Patterns** are a set of patterns that focus on how objects distribute responsibilities among them.
- **Behavioral Patterns** are used when you want to provide a standard way to iterate over a collection, save and restore the previous state of an object, define a family of algorithms, alter an object's behavior when its internal state changes, reduce chaotic dependencies between objects, turn a request into a stand-alone object, define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing. ✓
- **Behavioral Patterns** are not recommended when you have a system that doesn't change, or you have a system that doesn't have a lot of objects.

Conclusions

- **Behavioral Patterns** are a set of patterns that focus on how objects distribute responsibilities among them.
- **Behavioral Patterns** are used when you want to provide a standard way to iterate over a collection, save and restore the previous state of an object, define a family of algorithms, alter an object's behavior when its internal state changes, reduce chaotic dependencies between objects, turn a request into a stand-alone object, define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.
- **Behavioral Patterns** are not recommended when you have a system that doesn't change, or you have a system that doesn't have a lot of objects.

Outline

Thanks!

Questions?

Repo: *https://github.com/EngAndres/ud-public/tree/main/courses/software-modeling*