Principles and Patterns of Software Design

# Chapter One (Strategy Pattern)

SOLID Design Principles

* Single Responsibility Principle
  + A class should only have a single responsibility
  + Each method is responsible for one thing and one thing only
* Open-Closed Principle
  + Entities should be open for extension but closed for modification
* Liskov Substitution Principle
  + Objects should be replaceable with instances of their subtypes without altering program correctness.
* Interface Segregation Principle
  + Many client-specific interfaces better than one general-purpose interface
* Dependency Inversion Principle
  + Dependencies should be abstract rather than concrete

Pyramid of Joy (Good Design)

* Good Design
  + Maintainability
  + Extensibility
  + Scalability
  + Testability
  + Reusability
* Design Patterns
  + Singleton
  + Abstract Factory
  + DAO
  + Strategy
  + Decorator
* Design Principles
  + Program to an interface, not an implementation
    - Avoid hard coding
    - Have flexible code
  + High cohesion
    - Everything in the class should support a central purpose
  + Low coupling
    - Two modules can interact with each other without having to be concerned with how the other one’s code works
  + Open-Closed
  + Separation of concerns

Strategy Pattern Design Principle #1

* Identify the aspects of your application that vary and separate them from what stays the same
  + This principle underlies most patterns
* Encapsulate what varies

Strategy Pattern Design Principle #2

* Program to an interface, not an implementation

Composition

* “Has a” relationship
* A room and a house
  + A room and a house need each other in order to exist

The Strategy Pattern

* Defines a family of algorithms, encapsulates each one, and makes them interchangeable
* Strategy lets the algorithm vary independently from the clients that us it

Principles in Strategy

* Identify the aspects of your application that vary and separate them from what stays the same
* Program to an interface, not an implementation
* Favor composition over inheritance

Association

* Graphical user interface, text, application, email

  Description automatically generated
* Diagram

  Description automatically generated

Aggregation/Composition

* Text

  Description automatically generated

# Chapter Two (Observer Pattern)

Relative Path

* Do not hard code file paths
* Do a relative path instead (look into what the language offers)

Observer Pattern

* Parts of code (Observers) that are looking at another section of code (observable) and waiting for it to change state
* Useful for event driven designs
* Ex.
* Diagram

  Description automatically generated

Observer

* An observer is an object that wishes to be informed about events happening in the system, typically by providing a callback function call when events occur

Observerable

* The object that contains data that observers want in real time
* Observerable: “When something changes I’ll let you know”

Observer: “Thank you”

* Two different models for notifying observers
  + Push model: send data to observers regardless of whether they need it or not
  + Pull model: send notification to tell observers to pull whatever data they need

Graphical user interface, text, application

Description automatically generated

Example Code

* Driver

Graphical user interface, text, application

Description automatically generated

* Subject

Graphical user interface, application

Description automatically generated

* + Has to be either an abstract class or an interface
* Observerable

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

* Observer interface

Graphical user interface, text, application, email

Description automatically generated

* Observer objects

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Check List

Text, letter

Description automatically generated

# Chapter Three (Decorator Pattern)

Decorator Pattern UML

Diagram

Description automatically generated

Decorator Pattern Definition

* Attaches additional responsibilities to an object dynamically. Provides a flexible alternative to subclassing for extending functionality.

# Chapter Four (Factory Pattern)

Basic Kinds of Design Patterns

* Structural
  + Decorator
* Creational
  + Factory
  + Singleton
* Behavioral
  + Observer

Definition

* Defines an interface for creating an object, but lets subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses

# Chapter Five (Singleton Pattern)

Static

* Initializes the method/field at the start of the program

Singleton Pattern

* MAKE THE CONSTRUCTOR PRIVATE
* Ensures that a class has only one instance and provides a global access point to it.
* Make the class itself responsible for keeping track of its sole instance
* “There can only be one.”

Uses

* Only one instance of a WindowManager
* Only one instance of a filesystem

Thread Safe

* Singleton needs to be thread safe.
* One way is to create a static instance of it in the class itself.
  + Simple way
* A second way is to use the synchronized keyword in the getInstance() method
  + Lazy way, very expensive
* A third way is to use volatile keyword and synchronized keyword.
  + Double checking before going through the expensive process.
  + Ideal

Secondary Property

* Lazy loading
* Only call it when you need it

# Chapter Six (Command Pattern)

Command Pattern

* Encapsulates a command as an object, letting you parameterize clients with different requests – queue or log – requests and support undoable operations

Concepts

* Encapsulates request as an object
* Object-oriented callback
* Decouple sender from processor
* Often used for “undo” functionality

Design

* Object per Command
* Command interface
  + Execute Method
  + Unexecute Method (sometimes)
* Can use Reflection
  + Can call itself

Components

* Command
* Invoker
* ConcreteCommand

UML

Diagram

Description automatically generated

* Could have multiple receivers and ConcreteCommands

# Chapter Seven (Adapter Pattern)

Pattern Type

* Structural pattern

Highlighted Design Principles

* Open Closed Principle
* Dependency Inversion Principle

Intent

* Future proof
* Converts the interface of a class into another interface that clients expect
* Let classes work together that otherwise couldn’t

UML  
Diagram

Description automatically generated

# Chapter Eight (Façade Pattern)

Description

* Provides a unified interface to a set of interfaces in a subsystem
* Defines a higher-level interface that makes the subsystem easier to use
* Provides a simple, purpose-built interface to a larger, more complex body of code or interfaces

Design

* Shouldn’t use inheritance
* Utilizes composition
* Structural and re-factoring pattern
* No real UML

UML

* Diagram

  Description automatically generated
* Diagram

  Description automatically generated
* Diagram

  Description automatically generated

Consequences

* The use of existing API is simplified
* The façade will need to be updated (or a new one created) to expose more functionality from the sub-system

Façade vs Adapter

* Façade defines a new interface
* Adapter uses an old interface
* Adapter makes two existing interfaces work together as opposed to defining a new one

# Chapter Nine (Template Pattern)

Check list

* Examine the algorithm, and decide which steps are standard and which steps are peculiar to each of the current classes
* Define a new abstract base class to host t

# Chapter Ten (Composite/Collection Pattern & Iterative Pattern)

Design Principle

* Separate what varies and encapsulate it

Iterator Definition

* Provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation
* Provides a way to access and iterate through collections in the same way, regardless of type.

Iterator Pattern

* Ask object for its iterator
* Use the iterator to iterate through the items in the aggregate
* Iteration code works with any kind of aggregate object
* Requires you to write an iterator for each object that works with the collection type that the object uses

Iterator UML

* Diagram

  Description automatically generated

Iterator Concepts

* Traverse a container
* Doesn’t expose underlying structure
* Decouples algorithms
* Sequential

# Chapter Eleven (Visitor Pattern)

Visitor Pattern

* Behavior can be added to existing hierarchy
* Behavior is class specific

Design

* Interface based
  + Visitor and Element
  + Elements have visit method
* Modify each concrete class

UML

* Diagram

  Description automatically generated