Design Pattern

Patterns in Software

- "Designing object-oriented software is hard and designing reusable object-oriented software is even harder." - Erich Gamma
- Experienced designers reuse solutions that have worked in the past
- Well-structured object-oriented systems have recurring patterns of classes and objects
- Knowledge of the patterns that have worked in the past allows a designer to be more productive and the resulting designs to be more flexible and reusable

Design Patterns

- Design patterns describe the relations and interactions of different class or objects or types.
- They do not specify the final class or types that will be used in any software code, but give an abstract view of the solution.
- Patterns show us how to build systems with good object oriented design qualities by reusing successful designs and architectures.
- Expressing proven techniques speed up the development process and make the design patterns, more accessible to developers of new system.

Design Pattern

- In software engineering, a design pattern is a general reusable solution to a commonly occurring problem in software design.
- A design pattern is not a finished design that can be transformed directly into code.
- It is a description or template for how to solve a problem that can be used in many different situations.
- Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved.

Design Pattern

A good pattern should

  - Be as general as possible
  - Contain a solution that has been proven to effectively solve the problem in the indicated context.

- They give the developer a selection of tried and tested solutions to work with.
- They are language neutral and so can be applied to any languages that supports object orientation.
- They have a proven track record as they are already widely used and thus reduce the technical risk to the project.
- They are highly flexible and can be used in practically any type of application or domain.

The Originator of Patterns

- "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice." -- Christopher Alexander 1977
- “Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution.”

Classification of Design Patterns

- Design patterns were originally classified into three types
  - Creational patterns
  - Structural patterns
  - Behavioural patterns.

Creational Patterns

- Creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.
- The basic form of object creation could result in design problems or added complexity to the design.
- Creational design patterns solve this problem by somehow controlling this object creation.

Structural Patterns

- Structural design patterns are design patterns that ease the design by identifying a simple way to realise relationships between entities.
- These describe how objects and classes combine themselves to form a large structure

Behavioural Patterns

- Design patterns that identify common communication patterns between objects and realize these patterns.
- These patterns increase flexibility in carrying out this communication.

Structure of a Design Pattern

- Design pattern documentation is highly structured.
- The patterns are documented from a template that identifies the information needed to understand the software problem and the solution in terms of the relationships between the classes and objects necessary to implement the solution.
- There is no uniform agreement within the design pattern community on how to describe a pattern template.

Pattern Documentation

- **Pattern Name and Classification:** A descriptive and unique name that helps in identifying and referring to the pattern.
- **Intent:** A description of the goal behind the pattern and the reason for using it.
- **Also Known As:** Other names for the pattern.
- **Motivation (Forces):** A scenario consisting of a problem and a context in which this pattern can be used.
- **Applicability:** Situations in which this pattern is usable; the context for the pattern.
- **Structure:** A graphical representation of the pattern. Class diagrams and Interaction diagrams may be used for this purpose.
- **Participants:** A listing of the classes and objects used in the pattern and their roles in the design.

Pattern Documentation - cont.

- **Collaboration:** A description of how classes and objects used in the pattern interact with each other.
- **Consequences:** A description of the results, side effects, and trade offs caused by using the pattern.
- **Implementation:** A description of an implementation of the pattern; the solution part of the pattern.
- **Sample Code:** An illustration of how the pattern can be used in a programming language
- **Known Uses:** Examples of real usages of the pattern.
- **Related Patterns:** Other patterns that have some relationship with the pattern; discussion of the differences between the pattern and similar patterns.

Creational Patterns

- Abstract Factory - Creates an instance of several families of classes
- Builder - Separates object construction from its representation
- Factory Method - Creates an instance of several derived classes
- Prototype - A fully initialized instance to be copied or cloned
- Singleton - A class of which only a single instance can exist

Structural Patterns

- Adapter - Match interfaces of different classes
- Bridge - Separates an object’s interface from its implementation
- Composite - A tree structure of simple and composite objects
- Decorator - Add responsibilities to objects dynamically
- Facade - A single class that represents an entire subsystem
- Flyweight - A fine-grained instance used for efficient sharing
- Proxy - An object representing another object

Behavioral Patterns

- Chain of Resp. - A way of passing a request between a chain of objects
- Command - Encapsulate a command request as an object   Interpreter   A way to include language elements in a program
- Iterator - Sequentially access the elements of a collection
- Mediator - Defines simplified communication between classes
- Memento - Capture and restore an object's internal state
- Observer - A way of notifying change to a number of classes
- State - Alter an object's behavior when its state changes
- Strategy - Encapsulates an algorithm inside a class
- Template Method - Defer the exact steps of an algorithm to a subclass
- Visitor - Defines a new operation to a class without change