Day-13

Java Design Patterns

Agenda

- What are design patterns
- Why do we need design patterns
- Structure of design patterns
- Types of design patterns
- Overview of design patterns

What are design patterns

- In software engineering, a **design pattern** is a general repeatable solution to a commonly occurring problem in software **design**.

- A **design pattern** isn't 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.

- A design patterns are **well-proved solution** for solving the specific problem/task.
- A design pattern provides a general reusable solution for the common problems that occur in software design.
- The pattern typically shows relationships and interactions between classes or objects.
- The idea is to speed up the development process by providing well tested, proven development/design paradigms.
- Design patterns are programming language independent strategies for solving a common problem. That means a design pattern represents an idea, not a particular implementation.
- By using design patterns, you can make your code more flexible, reusable, and maintainable.

- Design Patterns are already defined and provides industry standard approach to solve a recurring problem, so it saves time if we sensibly use the design pattern. There are many java design patterns that we can use in our java based projects.

- Using design patterns promotes reusability that leads to more robust and highly maintainable code. It helps in reducing total cost of ownership (TCO) of the software product.

- Since design patterns are already defined, it makes our code easy to understand and debug. It leads to faster development and new members of team understand it easily.

Why do we need design patterns

- Design Patterns provide easy to recognize and use OOP solutions to common problems. They're inherently easy to maintain, because many people are familiar with them.

- This is very similar to how google works. Everyone knows HOW to google, so when you get a query like "What is the purpose of design patterns", you can very quickly use this common interface to solve a problem.

- It’s not mandatory to always implement design patterns in your project. Design patterns are not meant for project development. Design patterns are meant for common problem-solving. Whenever there is a need, you have to implement a suitable pattern to avoid such problems in the future.

Structure of design patterns

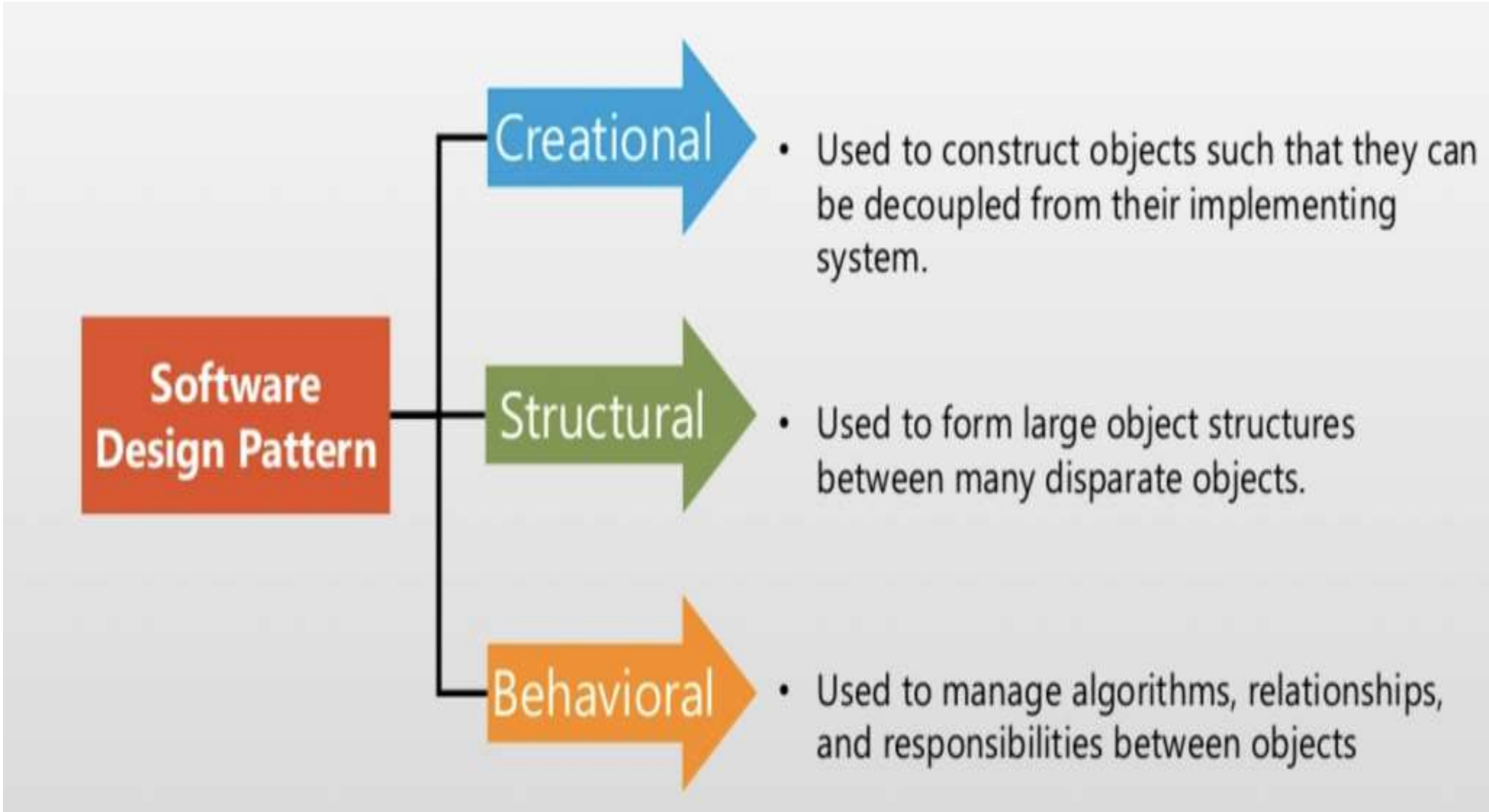
- Structure of design patterns can defined as graphical representation of the class involved in the pattern following the notations of object Modeling Technique.

Term	Description
Pattern Name	Describes the essence of the pattern in a short, but expressive name.
Intent	Describes what the Pattern Does
Also, known as	List any synonyms of the pattern
Motivation	Provides an example of the problem and describes the solution to it
Applicability	Lists the situations where the pattern is applicable
Structure	Set of diagrams of the classes and objects that depict the pattern
Participants	Describes the classes and objects that participate in the design
Collaborations	Describes the Participants collaboration and their responsibilities.
Consequences	Describes the forces that exist with the pattern and the benefits

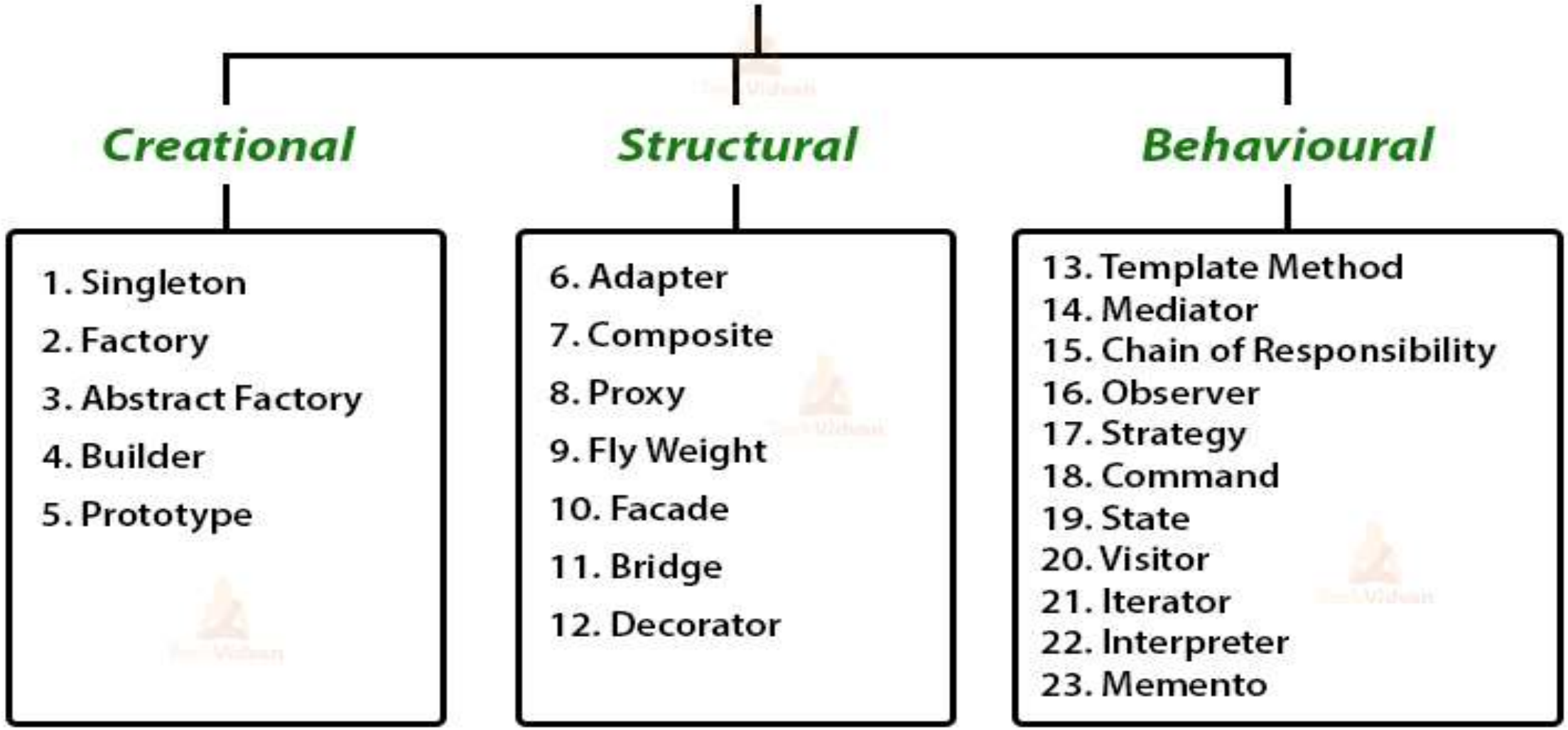
Types of design patterns

- Basically, design patterns are categorized into two parts:
  1. Core Java (or JSE) Design Patterns.
  2. JEE Design Patterns.

- There are about 26 Patterns currently discovered, These 26 can be classified into 3 types which belongs to core Java Design Patterns:-
  - **Creational**
  - **Structural**
  - **Behavioural**


Software Design Pattern
Creational
Used to construct objects such that they can be decoupled from their implementing system.
Structural
Used to form large object structures between many disparate objects.
Behavioral
Used to manage algorithms, relationships, and responsibilities between objects

Types of Design Pattern in Java



Creational design patterns

- Creational Patterns are design patterns that focus on how we obtain instances of objects.

- Typically, this means how we construct new instances of a class, but in some cases, it means obtaining an already constructed instance ready for us to use.

- In software engineering, creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.

- Creational design patterns are composed of two dominant ideas. One is encapsulating knowledge about which concrete classes the system uses. Another is hiding how instances of these concrete classes are created and combined.

- These patterns are designed for class instantiation. They can be either class-creation patterns or object-creational patterns.
- Creational design patterns provide solution to instantiate a object in the best possible way for specific situations.
- These design patterns are all about class instantiation.
- This pattern can be further divided into class-creation patterns and object-creational patterns.
- class-creation patterns use inheritance effectively in the instantiation process.
- object-creation patterns use delegation effectively to get the job done.

which is further divided into their sub-parts:

- **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
- **Object Pool**:- Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- **Prototype**:- A fully initialized instance to be copied or cloned
- **Singleton**:-A class of which only a single instance can exist

Structural design patterns

- Structural Patterns are concerned about providing solutions and efficient standards regarding class compositions and object structures.

- Also, they rely on the concept of inheritance and interfaces to allow multiple objects or classes to work together and form a single working whole.

- These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

- **Structural design patterns** are concerned with how classes and objects can be composed, to form larger structures.

- The structural design patterns **simplifies the structure by identifying the relationships**.

- These patterns focus on, how the classes inherit from each other and how they are composed from other classes.

- Structural design patterns show you how to glue different pieces of a system together in a flexible and extensible fashion. They help you guarantee that when one of the parts changes, the entire structure does not need to change.

- Adapter Pattern
  - Adapting an interface into another according to client expectation.

- Bridge Pattern
  - Separating abstraction (interface) from implementation.

- Composite Pattern
  - Allowing clients to operate on hierarchy of objects.

- Decorator Pattern
  - Adding functionality to an object dynamically.

- Facade Pattern
  - Providing an interface to a set of interfaces.

- Flyweight Pattern
  - Reusing an object by sharing it.

- proxy Pattern
  - Representing another object.

Behavioural Design Patterns

- Behavioural design patterns are concerned with algorithms and the assignment of responsibilities between objects.

- Behavioural design patterns are concerned with the interaction and responsibility of objects.

- In these design patterns, the interaction between the objects should be in such a way that they can easily talk to each other and still should be loosely coupled.

- Behavioural Patterns are concerned with providing solutions regarding object interaction - how they communicate, how are some dependent on others, and how to segregate them to be both dependent and independent and provide both flexibility and testing capabilities.

- Behavioral patterns provide solution for the better interaction between objects and how to provide lose coupling and flexibility to extend easily.
  - Chain Of Responsibility Pattern
  - Command Pattern
  - Interpreter Pattern
  - Iterator Pattern
  - Mediator Pattern
  - Memento Pattern
  - Observer Pattern
  - State Pattern
  - Strategy Pattern
  - Template Pattern
  - Visitor Pattern