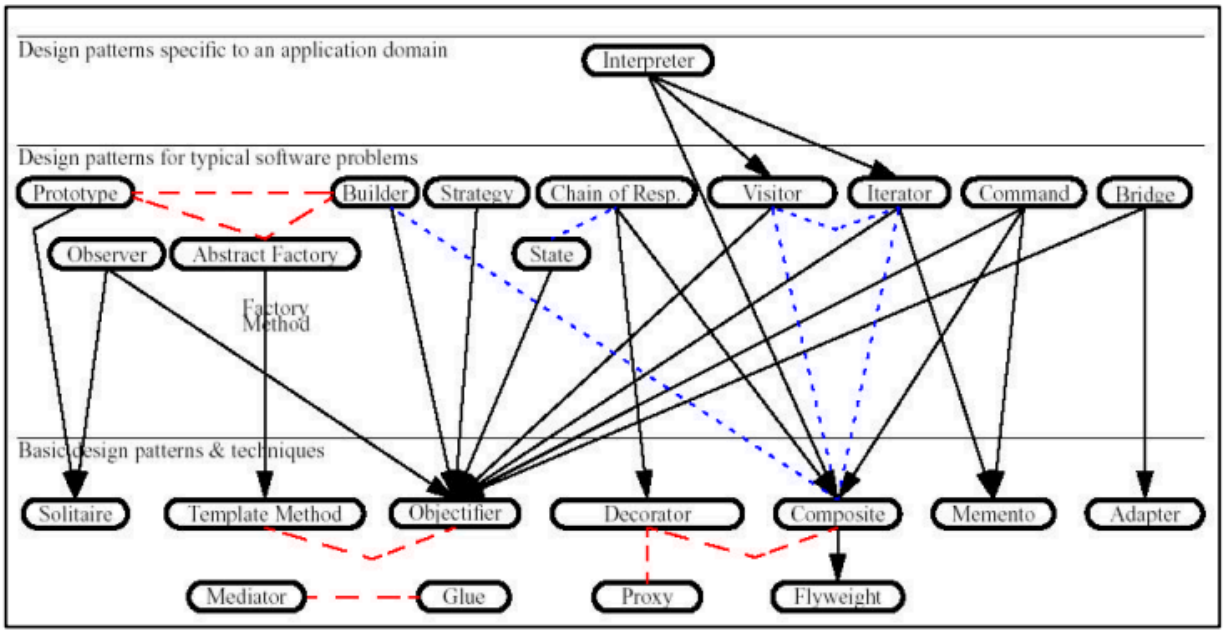
In "Design Pattern," **"Design"** refers to the way you plan and create software structures. It's like the blueprint for building something. **"Pattern"** means a repeatable and proven solution to a common problem. Design patterns are like guides for building software, based on past experiences. They help developers communicate better and reuse successful designs. By using design patterns, you can make systems more reusable and avoid choices that hurt reusability. They also make it easier to document and maintain software by specifying how classes and objects interact.

- **Categories of Design Patterns**
    1) **Design patterns can be classified by two criteria.** The first criterion, called **purpose** that reflects what a pattern does, divided design patterns into creational, structural, or behavioral purpose. **Creational patterns** such as abstract factory, builder, factory method, prototype and singleton concern the process of object creation. **Structural patterns** such as adapter, bridge, composite, decorator, facade, flyweight and proxy deal with the composition of classes or objects. **Behavioral patterns** such as chain of responsibility, command, interpreter, iterator, mediator, memento, observer, state, strategy, template method and visitor characterize the ways in which classes or objects interact and distribute responsibility.
    2) The second criterion of classification partitions design patterns into three semantically different layers: **Basic design patterns and techniques, design patterns for typical software problems(middle layer), design patterns specific to an application domain(high layer)**. Basic design patterns and techniques layer contains the design patterns, which are heavily used in the design patterns of higher layers and in object-oriented systems in general . The problems addressed by these design patterns occur again and again when developing object-oriented systems. The design patterns are thus very general. When building a system, one would often look upon them more as basic design techniques than as patterns. The intentions of these design patterns(Table 1) are very general and applicable to a broad range of problems occurring in the design of object-oriented systems.

Design pattern	Purpose of the design pattern contexts
Adapter	Adapting a protocol of one class to the protocol of another class
Composite	Single and multiple, recursively composed objects can be accessed by the same protocol
Glue	Encapsulating a subsystem
Mediator	Managing collaboration between objects.
Memento	Encapsulating a snapshot of the internal state of an object
Objectifier	Objectifying behaviour
Proxy	Attaching additional properties to objects
Solitaire	Providing unique access to services or variables
Template Method	Objectifying behaviour (primitives will be varied in subclasses)

Table 1. Basic design patterns with their respective purposes



Arrangement of design pattern in layers

Creational design patterns are concerned with the creation of objects. They provide ways to create objects in a manner that is suitable for the situation at hand, without specifying the exact details of how the objects are created.

Examples -
Prototype : A fully initialized instance to be copied or cloned
Example : A bakery that uses a prototype cake to create new cakes.

- java.lang.Object#clone()

Builder - Separates the construction of a complex object from its representation so that the same construction process can create different representations.
Example : A software development team that creates new software using a variety of libraries and frameworks.

- java.lang.StringBuilder

Singleton - A class of which only a single instance can exist
Example : President of a country

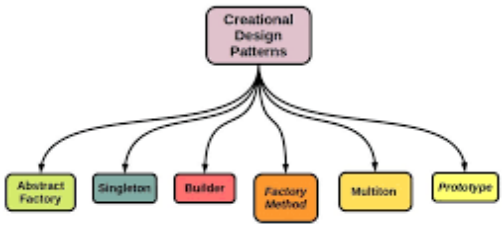
- java.lang.Runtime#getRuntime()

Factory Method - Creates a **f**amily of object types.
Example : In an organisation HR works as factory method. Here development team request type of resource need to HR. Based on request type, HR provide resource to Development team.

- java.util.Calendar#getInstance()

Abstract Factory - Creates an instance of several families of classes
Example : HP, Samsung and Dell laptops are uses Intel and AMD processor.

- javax.xml.parsers.DocumentBuilderFactory#newInstance()



Structural design patterns are concerned with the composition of classes and objects. They provide ways to combine objects and classes in a way that is flexible and efficient.

Examples -
Proxy - Introduces a proxy layer between clients and services to add functionalities like caching or security.

- java.rmi.*, the whole API actually.

Composite - Gives an unified interface to a leaf and composite. Compose objects into tree structures to represent part-whole hierarchies i.e. an object represt as part and also represent whole.
Example : File System in Operating Systems, Directories are composite and files are leaves. System call Open is single interface for both composite and leaf.

Decorator - Decorator design pattern is used to add the functionality by wrapping another class around the core class without modifying the core class.
Example : 1) Adding **d**iscounts on an order 2) gun is a deadly weapon on it's own. But you can apply certain "decorations" to make it more accurate, silent and devastating.

- All subclasses of java.io.InputStream, OutputStream, Reader and Writer have a constructor taking an instance of same type.

Facade - Provide a unified interface to a set of interfaces in a subsystem, making the subsystem easier to use.
Example : Control Panel, Event Manager.

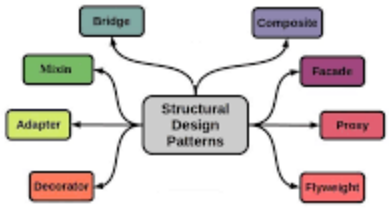
- javax.faces.context.ExternalContext, which internally uses ServletContext, HttpSession, HttpServletRequest, HttpServletResponse, et

Adapter - Adapter pattern is used when two unrelated interfaces need to work together.
Example : Power Adapters

- java.util.Arrays#asList()

Flyweight - A space optimization technique that lets us use less memory by sharing data.
Example : A software development team that uses a pool of database connections to reduce the overhead of creating and closing new connections.

- java.lang.Integer#valueOf(int) (also on Boolean, Byte, Character, Short and Long)



Behavioral design patterns are concerned with the communication between objects. They provide ways to define the communication between objects, as well as the flow of information between them.

Examples -
Chain of Responsibility is used to pass a request along a chain of handlers.
Example : Loan or Leave approval process, Exception handling in Java.

- javax.servlet.Filter#doFilter()

Iterator - Traverse/Sequentially access the elements of a collection
Example : A for loop in a programming language

- All implementations of java.util.Iterator & java.util.Enumeration

State - Allow an object to alter its behavior when its internal state changes
Example : A traffic light that cycles through the states of red, yellow, and green.
A vending machine that transitions through the states of idle, accepting money, dispensing product, and out of stock.
A video game character that can be in the states of standing, walking, running, and jumping.

Observer - A one-to-many dependency between objects, so that when one object changes state, all its dependents are notified and updated automatically
Example : A social media platform

- Publish/Subscribe JMS API

Visitor - Visitor pattern is used to add methods to different types of classes without altering those classes. Example : A database that performs different operations on different types of data. A web browser that renders different types of HTML elements.

Template - Defines a skeleton of an algorithm in an operation, and defers some steps to subclasses.
Example : A cooking recipe that provides a step-by-step template for preparing a dish. A software development lifecycle that provides a template for developing and delivering software.

- All non-abstract methods of java.io.InputStream, java.io.OutputStream, java.io.Reader and java.io.Writer.
- All non-abstract methods of java.util.AbstractList, java.util.AbstractSet and java.util.AbstractMap.
- javax.servlet.http.HttpServlet, all the doXXX() methods by default sends a HTTP 405 "Method Not Allowed" error to the response. You're free to implement none or any of them.
- JMSTemplate HibernateTemplate and JdbcTemplate in Spring

Command - Command pattern is used when request is wrapped and passed to invoker which then inturn invokes the encapsulated command.
Example : A remote control

- All implementations of java.lang.Runnable

Memento - Memento pattern is used to restore state of an object to a previous state.
Example : save the state in a game & Undo/Redo operation in Windows, A text editor that allows users to undo and redo their actions. A database that allows users to rollback transactions.

- All implementations of java.io.Serializable

Mediator - Mediator pattern is used to provide a centralized communication medium between different objects.
Example : Air Traffic Controller(ATC), Stock Exhange

Strategy - Strategy pattern is used when we have multiple algorithm for a specific task and client decides the actual implementation to be used at runtime.
Example : Modes of transportation

- java.util.Comparator#compare(), executed by among others Collections#sort().
- javax.servlet.http.HttpServlet, the service() and all doXXX() methods take HttpServletRequest and HttpServletResponse and the implementor has to process them (and not to get hold of them as instance variables!).
- javax.servlet.Filter#doFilter()

In summary, creational design patterns focus on object creation, structural design patterns focus on the composition of objects and classes, and behavioral design patterns focus on the communication between objects.

- **Comparative study of Design Patterns :**

Design Pattern Type	Purpose	Example Analogy	Key Patterns	Benefits
Creational	Object creation mechanisms	Recipes for specific dishes	Singleton, Factory Method, Abstract Factory, Builder	Flexibility and control in object creation
Structural	Composition of classes/objects	Building blocks forming structures	Adapter, Decorator, Composite, Bridge	Organizing classes for scalability and efficiency
Behavioral	Object interaction	Rules of interaction in a game or sport	Observer, Strategy, Command, State	Loose coupling and flexibility in interactions

References -

1)https://stackoverflow.com/questions/11553804/design-patterns-with-real-time-example#:~:text=Creational,while%20hiding%20the%20creation%20logic.&text=Example%20%3A%20A%20bakery%20that%20uses,cake%20to%20create%20new%20cakes.&text=Builder%20%2D%20Separates%20the%20construction%20of,process%20can%20create%20different%20representations

2)https://ecs.syr.edu/faculty/fawcett/Handouts/cse776/Lecture20/References/Design_patterns_Report.pdf .

Aarya Teli

B.Tech. (CSE)

KIT's College of Engineering (Autonomous) Kolhapur