Design Pattern in Java:

- We use design pattern for during the analysis and requirement phase of SDLC.

Creation Design Pattern:

- It is concerned with creating a object of class.
- It is used when a decision must be made at the time of instantiation of class.
- Type of :
    - Factory Method Pattern
    - Abstract Factory Pattern
    - Singleton Pattern
    - Prototype Pattern
    - Builder Pattern
    - Object Pool Pattern

1. Factory Method Pattern:

- It is used define an interface or abstract class for creation object but let the subclass decide which class to instantiate.
- Subclass is responsible for create instance of class.
- It is also known as virtual constructor.

➔Advantage:

- It's allow to choose sub-class for type of object to create.
- It promotes the loose coupling, bind the specific class into code.

➔Uses:

- When a class does not known the what sub-class will be required to create.
- When a class want that its sub-class specify the object to be created.
- When a parent class choose the creation of object to its sub-class.

2. Abstract Factory Pattern:

- It is used to define interface and abstract class for creating familiar of related object but without concrete sub-classes.
- It's return the factory of classes.
- It is also known as kit.

➔Advantage:

- It is used to isolate the client code from concrete(implementation class) classes.
- It eases the exchanging of object families.
- It promotes consistency among object.

➔Uses:

- When system needs to be independent of how its object is created, compose and represent.
- When the family of related object is used together, then constraint need to enforced.
- When you want to provide a library of object that does not show implementations and only reveals interfaces.
- When system need to configured with one of a multiple family of object.

3. Singleton Pattern:

- Define a class that has only one instance and provides a global point to access to it.
- Single object is create only one and used by all other classes.
- There are two forms of singleton pattern
    - Early instantiation : creation of instance at load time.
    - Lazy instantiation : creation of instance when required.

➔Advantage:

- Save memory

➔Use:

- It mostly used in multi-thread and database application.

➔How to create:

- To create, we need to have static member of class, private constructor and static factory method.
- Static factory method : it's provide the global point to access to the singleton object and return the instance to the caller.

4. Prototype Design Pattern:

- Cloning of an existing object instead of creating new one and also be customized as per the requirement.
- If the cost of creating a new object is expensive and resource intensive.

➔Advantage:

- It reduce the need of sub-classing.
- It hides complexities of creating object.
- The client get new object without knowing which type of object it will be.
- It lets you add and remove objects at runtime.

➔Uses:

- When the class are instantiated at runtime
- When the cost of creating an object is expensive or complicated.
- When you want to keep the number of classes in an application minimum.
- When the client application need to be unaware of object creation and representation.

5. Builder Design Pattern:

- Construct a complex object from simple object using step-by-step approach.
- It is mostly used when object can not be created in single step like in the de-serialization of a complex object.

➔Advantage:

- It provides clear separation between the construction and representation of an object.
- It provides better control over construction process.
- It support to change the internal representation of objects.

6. Object Pool Pattern:

- To reuse the object that are expensive to create.
- It is container to contain specified amount of object.
- Object in the pool have a lifecycle :creation , validation and destroy.

➔Advantage:

- It boots the performance of the application significantly.
- It is most effective in a situation where the rate of initializing a class instance is high.
- It manages the connections and provides a way to reuse and share them.
- It can also provide the limit for maximum number of object that can be created.

➔Use:

- When an application requires object which are expensive to crate.
- Many connection for the database then it take too longer to create a new one and the database server will be overload.
- When there are several clients who need the same resource at different times.

Structural Design Pattern:

- It is concerned with how to object and class can be composed , to form larger structure.
- It simplifies the structural by identifying the relationships.
- Its focus on how to the classes inherit from each other and how they are composed from other classes.

Types:

- Adapter pattern
- Bridge pattern
- Composite pattern
- Decorator pattern
- Façade pattern
- Flyweight pattern
- Proxy pattern

1. Adapter pattern

- It is used for convert the interface into another interface that a client wants.
- To provide the interface according to the client requirement while using the services of a class with different interface.
- It is also known as wrapper class.

➔Advantage:

- It allows two or more previously incompatible objects to interact.
- It allows reusability of existing functionality.

➔Use:

- When an object needs to utilize an existing class with an incompatible interface.
- When you want to create a reusable class that cooperates with classes which don’t have compatible interface.
- When you want to create a reusable class that cooperates with classes which don’t have compatible interfaces.

2. Bridge Pattern:

- Decouple the functional abstraction from the implementation so that the two can vary independently.
- It is also known as handle or body.

➔Advantage:

- It enable the separation of implementation from the interface.
- It improves the extensibility.
- It allows the hiding of implementation details from the client.

➔Uses:

- When you don't want to permanent binding between the functional abstraction and its implementation.
- When both the functional abstraction and its implementation need to extended using sub-classes.
- It is mostly used in those places where change are made in the implementation does not affect the clients.

3. Composite Pattern:

- It allows to client to operate in generic manner on objects that may or may not represent a hierarchy of object.

➔Advantage:

- It define class interface that contain primitive and complex object.
- It makes easier to you to add new kinds of components.
- It provides flexibility of structure with manageable class or interface.

➔Uses:

- When you want to represent a full or partial hierarchy of objects.
- When the responsibilities are needed to be added dynamically to the individual objects without affecting other objects, where the responsibility of object may vary from time to time.

4. Decorator Pattern:

- Attach a flexible additional responsibilities to an object dynamically.
- It uses composition instead of inheritance to extend the functionality of an object at runtime.
- It is also known as wrapper.

➔Advantage:

- It provides the greater flexibility than static inheritance.
- It enhances the extensibility of the object , because changes are made by coding new classes.
- It simplifies the coding by allowing you to develop a series of functionality from targeted classes instead of coding all of the behaviour into the object.

➔Uses:

- When you want to transparently and dynamically and responsibilities to objects without affecting other objects.
- When you want to add responsibilities to an object that you may want to change in future.
- Extending functionality by sub-classing is no longer practical.

5. Façade Pattern:

- Just provides a unified and simplified interface to a set of interfaces in a subsystem therefore it hides the complexities of the subsystem from the client.
- It describe a high-level interface that makes the sub-system easier to use.
- Every abstract factory is a type of façade.

➔Advantage:

- It shields the clients from the complexities of the sub-system components.
- It promotes loose coupling between subsystem and its clients.

➔Uses:

- When you want to provides simple interface to complex sub-system.
- When several dependencies exits between clients and the implementation classes of an abstraction.

6. Flyweight Pattern:

- To reuse already existing similar kind of objects by storing them and create new object when no matching object is found.

➔Advantage:

- It reduces the number of objects.
- It reduces the amount of memory and storage devices required if the objects are persisted.

➔Uses:

- When an application use number of object.
- When the storage cost is high because of the quality of objects.
- When the application does not depend on object identity.

7. Proxy Pattern:

- It means object represent another object.
- Provides the control for accessing the original object.
- We can perform many operation like hiding the information of original object, on demand loading.
- It also known as surrogate of placeholder.

➔Advantage:

- It provides the protection to the original object from the outside world.

➔Uses:

- Virtual Proxy
- Protective Proxy
- Remote Proxy
- Smart Proxy

Behavioural Pattern:

- It is concerned with the interaction and responsibility of objects.
- The interaction between the objects should be in a such way that they can easily talk to each other and still should be loosely coupled.
- The implementation and the client should be loosely coupled in order to avoid hard coding and dependencies.

➔Types:

1. Chain of Responsibility Pattern:

    - In chain of responsibility , sender sends a request to a chain of objects.
    - The request can be handled by any object in the chain.
    - Avoid coupling the sender of a request to its receiver by giving multiple objects a chance to handle the request.
      E.g : ATM
    - Each receiver contains reference of another receiver. If one objects cannot handle the request then it passes the same to the next receiver and so on.

➔Advantage:

- It reduces the coupling.
- It adds flexibility while assigning the responsibilities to objects.
- It allows a set of classes to act as one; events produced in one class can be sent to other handler classes with help of composition.

➔Used:

- When more than one object can handle a request and the handler us unknown.
- When the group of object that can be handle the request must be specified in dynamic way.

2. Command Pattern:

- Encapsulate a request under an object as a command and pass it to invoker object.
- Invoker objects looks for the appropriate object which can handle this command and pass the command to the corresponding object and that object executes the command.
- It is also known as Action or Transaction.

➔Advantage:

- It separates the object that invokes the operation from the object that actually performs the operation.
- It makes easy to add new commands, because existing classes remain unchanged.

➔Use:

- When you need parametrize objects according to an action perform.
- When you need to create and execute request at different times.
- When you need to support rollback, logging or transition functionality.

3. Interpreter Pattern:

- To define a representation of grammar of a given language, along with an interpreter that used this representation to interpret sentences in the language.
- This pattern can applied for parsing the expression defined in simple grammars and sometimes in simple rule engines.

➔Advantage:

- It is easier to change and extend the grammar.
- Implementing the grammar is straight forward.

➔Use:

- When the grammar of the language is not complicated.
- When the efficiency is not a priority.

4.Iterator Pattern:

- To access the elements of an aggregate object sequentially without exposing its underlaying implementation.
- It is also known as Cursor.

➔Advantage:

- It supports variations in the traversal of a collection.
- It simplifies the interface to the collection.

➔Uses:

- When you want to access a collection of object without exposing its internal representation.
- When there are multiple traversals of object need to be supports in the collection.

5.Mediator Pattern:

- To define an objects that encapsulates how a set of objects interact.
- It used to reduce communication complexity between multiple objects or classes.
- This Pattern provides a mediator class which normally handles all the communication between different classes and supports easy maintainability of the code by loose coupling.

➔Benefits:

- It decouples the number of classes.
- It simplifies object protocols.
- It centralize the control.
- The individual components become simpler and much easier to deal with because they don’t need to pass message to one another.
- The components don’t need to contain logic to deal with their intercommunication and therefore they are more generic.

➔Use:

- It is commonly used in message-based systems likewise chat applications.

- When the set of objects communicate in complex but in well defines way.

6. Memento Pattern:

- To restore the state of an objects to its previous state.
- It must do this without violating encapsulation.
- It is also known as Token.
- It is used to implements the undo operation.
- This is done by saving the current state of the objects as it changes state.

→Benefits:

- It preserves encapsulation boundaries.
- It simplifies the originator.

➔Use:

- It is used in undo and redo operation in most software.
- It is also used in database transition.

7. Observer Pattern:

- Just define one to one dependency so that when one object changes state, all its depends are notified and update automatically.
- It is also known as Dependents or publish subscribe.

➔Benefit:

- It describe the coupling between the object and the observer.
- It provide the support for broadcast-type communication.

➔Use:

- When the change of a state in one object must be reflected in another object without keeping the objects tight coupled.
- When the framework we writes and need to be enhanced in future with new observes with minimal changes.

8. State Pattern:

- The class behaviour changes on its state.
- We create objects which represent various states and a context object whose behaviour varies as its state object changes.
- It is also known as Object for states.

➔Benefits:

- It keeps the state-specific behaviour.
- It makes any state transitions explicit.

➔Use:

- When the behaviour of object depends on its state and it must be able to change its behaviour at runtime according to the new state.
- It is used when the operations have large, multipart conditional statements that depend on the state of an object.

9. Strategy Pattern:

- Define a family of functionality , encapsulate each one , and make them interchangeable.
- It is also known as Policy.

➔Benefits:

- It provides a substitute to subclassing.
- It defines each behaviour within its own class, eliminating the need for conditional statements.
- It makes it easier to extend and incorporate new behaviour without changing the application.

➔Use:

- When the multiple classes differ only in their behaviours e.g: servlets API.
- It is used when you need different variations of an algorithm.

10.Template Pattern:

- Just define the skeleton of a function in an operation , deferring some steps to its subclasses.

➔Benefits:

- It is very common technique for reusing the code. This is only the main benefit of it.

➔Use:

- It is used when the common behaviour among sub-classes should be moved to a single common class by avoiding the duplication.