Software Design Patterns applied to Kotlin and Android development

What are design patterns

Solving similar problems in similar ways
Standard terminology
Best practices
Gang of Four

Types of design patterns
3 types

- Creational : handle creation of object
- Structural : how to structure a code
- Behavioural : object how communicate.

Creational Patterns
===================
1)Singleton:
  -only single instance
  -single point of access to the resources
 uses : network manager, database access, logging,   utility classes

2)Factory : factory method provide way to access functionality without caring about implementation
  -sepration of concern
  -allow for testability
3)Abstract Factory : factory method provide way to access functionality without caring about implementation
  -one level of abstraction over factory pattern
  -separation of concern
  -allow for testability
4)Builder : used when we have multiple parameters to initialise
  - for many parameters, it's impractical to build all   constructors
  -5 parameter combination -> 120 constructor
  -kotlin solve this problem partially with named param but does not work with java
5)Lazy Initialisation : very useful technique for memory management
why keep big object in memory if your not using it that is main idea behind it
  - initialise a resource when it needed not when it   declared
  -lazy vs eager initialisation
  -kotlin has build in lazy initialisation can only be used with val property . kotlin allow for late initialisation for var   properties crash if variable is not initialise before use.
6)Prototype : lets you copy existing objects, without depending on their classes, only reliant on interface,

The copy object must provide copy functionality

Structural Patterns
========================

1)Adapter : converts the interface of the class into another interface the client expect
-convert data from one format into another
-used extensively in android
2)Bridge: having classes with multiple orthogonal traits exponentially increases the size of the inheritance tree
-split into multiple interfaces / classes
-associate them using a bridge reference
3)Facade : provide simple interface to complex functionality
-removes need for complex object / memory management
e.g : retrofit
-simplified client implementation
4)Decorator/ Wrapper pattern : attach new behaviour to an object
-without altering existing code
-override existing behaviour
5)Composite : compose objects into tree structures
- works when the core functionality can be represented as a tree.
-manipulate many objects as single one
6)Proxy : Provide some functionality before and/or after calling an object
-similar to facade, except the proxy has same interface
-similar to decorator , except the proxy manages lifecycle of it's object

Behavioural Patterns
=========================

1)Observer : Defines a subscription mechanism
-Notify multiple objects simultaneously
-One to many relationship
2)Chain of Responsibilities : Define chain of handlers to process a request.
-it also help to achieve separation of concerns
-each handler contain reference to the next handler
-each handler decides to process the request And/ or pass it on
-request can be of different types
3)Command : A request is wrapped in an object that contains all request info
-The command object is passed to the correct handler

4)Strategy : A class behaviour or algorithm can be changed at runtime
-objects contain algorithm logic
-context object that can handle algorithm objects

-useful when we want to be able to add functionality without changing program structure

5)State : An object changes its behaviour based on an internal state

-at any moment , there is finite number of states a program can be in

-state can be encapsulated in an object

6)Visitor : Sepration between an algorithm an objects they operate on

-2 concepts visitor and element(visitable)

-the element accepts the visitor type objects

-Visitors performs the operation on the element objects.

7)Mediator : provide a central object used for communicating between objects

-Objects don't talk to each other

-Reduce dependencies between objects.

8)Memento : Save and restore the previous state without revealing implementation details

-3 Components :

1. Memento - stores the state 2. Originator - creates the state 3. CareTaker- decides to save or restore the state