Android Design Patterns

1. Model View Controller (MVC)

- Separate the modeling of the domain, the presentation, and the action based on user input into three separate classes:
    - **Model**
        - manage the behavior and data of the application domain
        - responds to requests for information about its state
        - responds to the instruction to change state
    - **View**
        - manage the display of the information / UI
    - **Controller**
        - interprets the mouse and keyboard input from the user
        - informs the model and / or the view to change as appropriate
- Implementation in Android
    - Model:
        - the classes required to implement the objects and object collections to handle database transactions. For example, the class of `Book` and a class of `BookCollections`
    - View:
        - displayed each book as widgets in the application
    - Controller:
        - the activity classes that receives cammands from the views

2. Observers / Observables

- Defines a one-to-many dependency between objects so that when one object change state, all its dependencies are notified and update automatically
- In MVC, this pattern is used to decouple models from views
- Implementation in Android
    - Button-click events
        - **Observer**: the `View.onClickListener` interface

        - **Concrete Observer**: the anonymous class or method that handles the event when the button is clicked
        - **Observables**: the button itself

3. Builder

- Defines an instance for creating an object but letting subclasses decide which class to instantiate
    - **Builder**:
        - specifies an abstract interface for creating part of a product object
    - **Concrete Builder**:
        - construct and put together parts of the product by implement the Builder interface
        - define and keep track of the representation it creates and provides the interface for saving the product
    - **Director**:
        - construct the complex object using the builder interface
    - **Product**:
        - represent the complex object that is being build
- Implementation in Android
    - `FragmentManager` for binding `Fragment` with frame layout widget

4. Factory Method

- Defines an interface for creating objects, but let subclasses to decide which class to instantiate
    - **Product**:
        - defines the interface for objects the factory method creates
    - **Concrete Product**:
        - implement the product
    - **Creator / Factory**
        - declares the method FactoryMethod which returns a Product object
        - may call the generating method for creating Product object
    - **Concrete Creator / Factory**

        - overrides the generating method for creating ConcreteProduct object
- Implementation in Android
    - Using `newInstance()` to instantiate a fragment

5. Adapter

- Convert the interface of a class into another interface client expect
- Adapter lets classes work together, that could not otherwise because of incompatible interface
- Implementation in Android
    - Recycler View Adapter

6. Decorator (Wrapper)

- Add additional responsibilities dynamically to an object
- Applies when there is a need to dynamically add as well as remove responsibilities to a class, and when subclassing would be impossible due to the large number of subclasses that could result
- Implementation in Android
    - SQLite `CursorWrapper`