**Design Parten**: tạo mẫu code chung để xử lý dữ liệu

Có 3 loại chính là : Khởi tạo (Builder, factory, sington, Dependence Injection), Cấu trúc(Adapter, Facade), Hành vi (Command, Observer, MVC, MVVM)

**I. Khởi tạo**

* Builder: Phân chia khởi tạo thành nhiều phần nhỏ, tạo theo từng đặc trưng của đối tượng

Vd: Notification, alertdialog

* Sington: chỉ tồn tại một instance duy nhất của một class nào đó trong toàn bộ chương trình
* DI: Cung cấp, khởi tạo nhanh 1 đối tượng khi cần tại vị trí bất kì trong ứng dụng bằng cách inject vào contructor thay vì tự tạo ra sự phụ thuộc
* chúng ta tạo instance của Class A trong Class B để thực hiện công việc mà A có thể làm. Với Dependence Injection(DI) thì instance của A sẽ được tạo ở bên ngoài B, và B nhận tham chiếu tới instance của A mà không phải tự khởi tạo nó.

Vd: dagger2, roboguice

* Factory: nhằm giải quyết vấn đề tạo một đối tượng mà không cần thiết chỉ ra một cách chính xác lớp nào sẽ được tạo. Factory method giải quyết vấn đề này bằng cách định nghĩa một phương thức cho việc tạo đối tượng, và các lớp con thừa kế có thể override để chỉ rõ đối tượng nào sẽ được tạo.

**II. Cấu trúc**

* Adapter : cho phép 2 lớp không tương thích làm việc được cùng nhau bằng cách chuyển đổi giao diện (interface) của một lớp sanggiao diện khác phù hợp với yêu cầu của khách hàng.
* Façade: cung cấp các high level interface để các interface khác dễ sử dụng hơn( Retrofit)

**III. Hành vi**

* Command: Đóng gói các request như là các object và gửi đi, việc thực hiện yêu cầu ntn sẽ đc thực hiện ở cơ chế khác
* Obsever: một phụ thuộc 1-n giữa các đối tượng. Khi một đối tượng thay đổi trạng thái, các đối tượng phụ thuộc nó sẽ được thông báo và tự động cập nhật.
* MVC Model(Các lớp dữ liệu) –View( Tương tác với người dùng) – Controler( Cập nhật view và update Model)
* MVVM: Model – View – View Model : View Model thực hiện các code logic ràng buộc view với model bằng bindingdata -> update runtime
* MVP: các logic được tách ra khỏi lớp trình bày, như vậy view sẽ được tách độc lập với model

**2. BindingData**

Liên kết dữ liệu logic với các UI Elements

**3. DI**

Khái niệm: tách lớp code

* RoboGuice: sử dụng Java Annotation, được nhúng bên trong code Java, để xác định những gì đã được inject ở đâu
* Dagger 2:

@Inject : (Có contructor)

. Sử dụng bởi 1 thành phần để thông báo rằng nó cần những phụ thuộc

. thông báo với Dagger sử dụng contructor này để khởi tạo đối tượng nếu muốn inject nó dưới dạng phụ thuộc

* Không có contructor( Retrofit, khởi tạo context):

@Provides: Đánh dấu 1 phương thức để báo cho Dagger biết phương thức này cung cấp dữ liệu trả về. Đối với cả hai kiểu trả về sẽ là Context. Dagger cung cấp [@Named](https://viblo.asia/u/Named) để đánh dấu Id cho các phương thức [@Provides](https://viblo.asia/u/Provides) của bạn. vd @Named(“ApplicationContex”)

[**@Module**](https://viblo.asia/u/Module) : định nghĩa các phương thức [@Provides](https://viblo.asia/u/Provides)  ở [@Module](https://viblo.asia/u/Module). Nó dùng trên một Class để nhóm các phương thức [@Provides](https://viblo.asia/u/Provides) có liên quan lại với nhau.

@Component định nghĩa  dùng những Module nào và nơi nó cần để inject

Định nghĩa cơ bản:

* [@Module](https://viblo.asia/u/Module) : cho các lớp mà có phương thức cung cấp sự phụ thuộc.
* [@Providers](https://viblo.asia/u/Providers) : cho các phương thức bên trong các lớp [@Module](https://viblo.asia/u/Module).
* [@Inject](https://viblo.asia/u/Inject) : để yêu cầu một sự phụ thuộc ( một hàm khởi tạo, một trường, hoặc một phương thức).
* [@Component](https://viblo.asia/u/Component) : là cầu nối interface giữa các modules và injection.

<https://viblo.asia/p/dependency-injection-voi-dagger-2-trong-android-Az45bAYwlxY>

* 2 cách để inject:
  + Thêm vào trong contructor
  + phụ thuộc được instance sau khi class được khởi tạo( trường hợp ko có contructor hoặc contructor tĩnh)