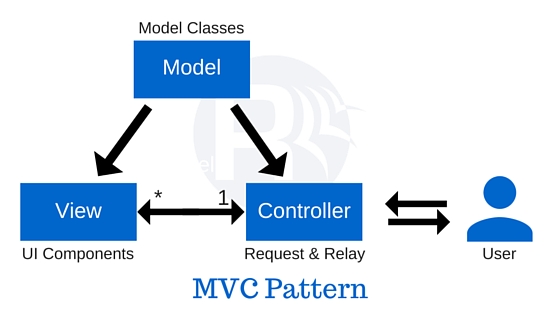
MVC:



**Model:**

Model bao gồm một tập hợp các class mô tả nghiệp vụ của ứng dụng, các đối tượng làm việc trong ứng dụng và các class truy cập dữ liệu. Nó cũng định nghĩa các quy tắc nghiệp vụ để dữ liệu có thể được sửa và thao tác ra sao.

**View:**

View hiển thị các thành phần giao diện người dùng như CSS, jQuery hay HTML…View hiển thị dữ liệu được nhận về từ controller như là đầu ra. Đây cũng là nơi thay đổi dữ liệu của Model do người dùng nhập vào.

### Controller

Controller có trách nhiệm xử lý các yêu cầu (request) được gửi đến. Nó sẽ xử lý các dữ liệu của người dùng qua Model và trả về kết quả ở View

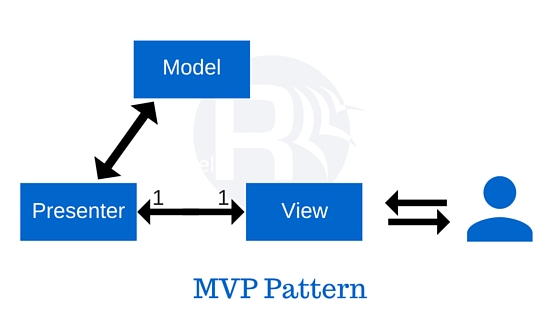
### Đánh giá

MVC rất tốt trong việc phân chia model và view. Chắc chắn sẽ dễ dàng test model vì nó không liên quan đến view và view không có gì nhiều để test (unit test). Tuy nhiên Controller vẫn còn nhiều hạn chế..

#### Mặt hạn chế của Controller

* Khả năng kiểm thử (test) - Controller bị ràng buộc với Android API nên sẽ khó để thực hiện unit test.
* Tính linh hoạt - Controller liên quan khá chặt chẽ với các view. Nếu chúng ta thay đổi view chúng ta sẽ phải thay đổi lại ở controller.
* Khả năng duy trì - Qua thời gian, controller sẽ ngày càng phình to ra do việc thêm code dẫn đến việc khó kiểm soát, View cũng thế

MVP (Presenter)

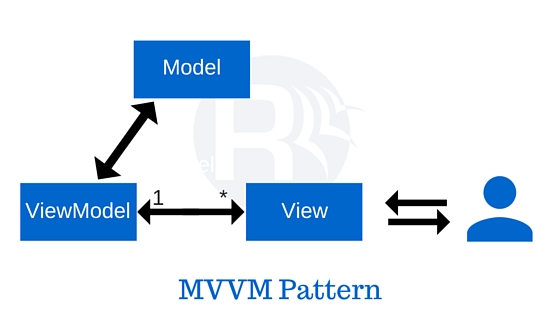


**Presenter có trách nhiệm quản lý tất cả các sự kiện từ giao diện thay cho View**. Nó nhận input từ người dùng thông qua View, sau đó xử lý dữ liệu thông qua Model. View và Presenter được hoàn toàn chia tách, giao tiếp với nhau qua thông qua interface, không giống View và Controller, mỗi khi có giao tiếp giữa 2 thành phần này bằng một interface**.** Presenter cũng không nhận các request vào giống Controller.

Presenter sẽ control view hiển thị những gì. Nói cách khác thì Prester sẽ là cầu nối giữa View và Model. **Trong MVP View sẽ không còn làm việc trực tiếp với Model giống như MVC.**

Hạn chế giống MVC

MVVM



ViewModel có trách nhiệm hiển thị các phương thức, các sự kiện và các hàm khác giúp cho việc bảo đảm trạng thái của View, việc thao tác trên Model là kết quả của các hành động trên View, và kích hoạt các sự kiện trên chính View.

**View-Model:**sẽ đảm nhận công việc đồng bộ dữ liệu từ model lên View. **Mối quan hệ giữa View và View-Model là View sẽ được ánh xạ tới View-Model nhưng View-Model lại không biết thông tin gì về View**

SOLID, MVC

* Design Patterns định nghĩa và cung cấp các tiếp cận chuẩn industry trong việc giải quyết các vấn đề có tính lặp lại. Nó giúp tiết kiệm thời gian khi chúng ta sử dụng một cách hợp lý theo hoàn cảnh phù hợp
* Sử dụng design pattern làm tăng **tính tái sử dụn**g, dẫn đến hiệu quả hơn trong maintain code, giúp giảm chi phí trong quá trình phát triển.
* Khi các design patterns được định nghĩa, nó làm cho code đơn giản, dễ hiểu , dễ debug hơn. Giúp phát triển phần mềm 1 cách nhanh chóng và các members trong team cũng dế dàng hiểu hơn.

3 design pattern nổi bật thuộc 3 nhóm khác nhau : Creational, Structure, Behavior

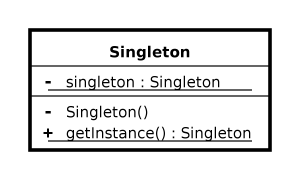
Creational:

* Singleton Pattern
* Factory Pattern
* Abstract Factory Pattern
* Builder Pattern
* Prototype Pattern

1. **Java Singleton** (Creational Design Patterns)

Singleton Pattern: một class duy nhất có trách nhiệm tạo ra một đối tượng trong khi đảm bảo rằng chỉ có một đối tượng được tạo ra. Class này cung cấp một cách để truy cập đối tượng duy nhất của nó mà có thể được truy cập trực tiếp mà không cần phải tạo ra 1 đối tượng mới khác để sử dụng

* **Contructor** private để giới hạn việc tạo đối tượng của class từ class khác.
* **Variable** private static trên cùng 1 class để chỉ có thể truy suất từ bản thân class đó.
* **Method** public static để trả về instance của class, đây chính là global access point ra thế giới bên ngoài



* singleton: Singleton -> biến private, kiểu dữ liệu là Singleton
* Singleton(): private constructor

+ getInstance(): method lấy biến instrance, kiểu dữ liệu Singleton

* **Eager initialization**

instance được tạo ra nhưng lại không được sử dụng

* public class EagerInitializedSingleton {
* private static final EagerInitializedSingleton instance = new EagerInitializedSingleton();
* //private constructor to avoid client applications to use constructor
* private EagerInitializedSingleton(){}
* public static EagerInitializedSingleton getInstance(){
* return instance;
* }
* }

#### Static block initialization

#### cấp thêm khả năng handle exception không trường hợp không thể new instance.

public class StaticBlockSingleton {

private static StaticBlockSingleton instance;

private StaticBlockSingleton(){}

//static block initialization for exception handling

static{

try{

instance = new StaticBlockSingleton();

}catch(Exception e){

throw new RuntimeException("Exception occured in creating singleton instance");

}

}

public static StaticBlockSingleton getInstance(){

return instance;

}

}

#### 3. Lazy Initialization

instance chỉ được tạo ra khi cần dùng tới.

Tại cùng 1 thời điểm 2 thread sẽ sinh ra đồng thời 2 instance khác nhau từ đó làm block singleton class.

public class LazyInitializedSingleton {

private static LazyInitializedSingleton instance;

private LazyInitializedSingleton(){}

public static LazyInitializedSingleton getInstance(){

if(instance == null){

instance = new LazyInitializedSingleton();

}

return instance;

}

}

1. **Adapter Pattern** (Structural Design Patterns)

Adapter Pattern thuộc vào nhóm mẫu thiết kế Cấu trúc, có chức năng làm cho 2 giao diện không liên quan tới nhau có thể làm việc cùng nhau. Nói đơn giản hơn, một chiếc điện thoại chạy pin 5V khi sạc điện vào ổ cắm 220V cần dùng 1 đốc sạc ( đây chính là Adapter).

Mình có 1 class CheckNumberAdaptee chuyên thực hiện việc kiểm tra 1 chuỗi ký tự có phải là kiểu số không.

Client bây giờ cần chức năng kiểm tra 1 chuỗi ký tự có phải là số điện không.

Rõ ràng client không thể dùng trực tiếp được CheckNumberAdaptee, ta sẽ dùng thêm class Adapter để thực hiện chức năng kiểm tra số điện thoại bằng cách sử dụng chức năng của CheckNumberAdaptee kiểm tra chuỗi ký tự đó có phải là kiểu số không, nếu đúng thì kiểm tra thêm điều kiện của số điện thoại.

