Trường ĐH Khoa Học Tự Nhiên Tp. Hồ Chí Minh
TRUNG TÂM TIN HỌC

Lập trình Android

Bài 33: *Mô hình Model – View - Presenter*

Phòng LT & Mạng

http://csc.edu.vn/lap-trinh-va-csdl

2019

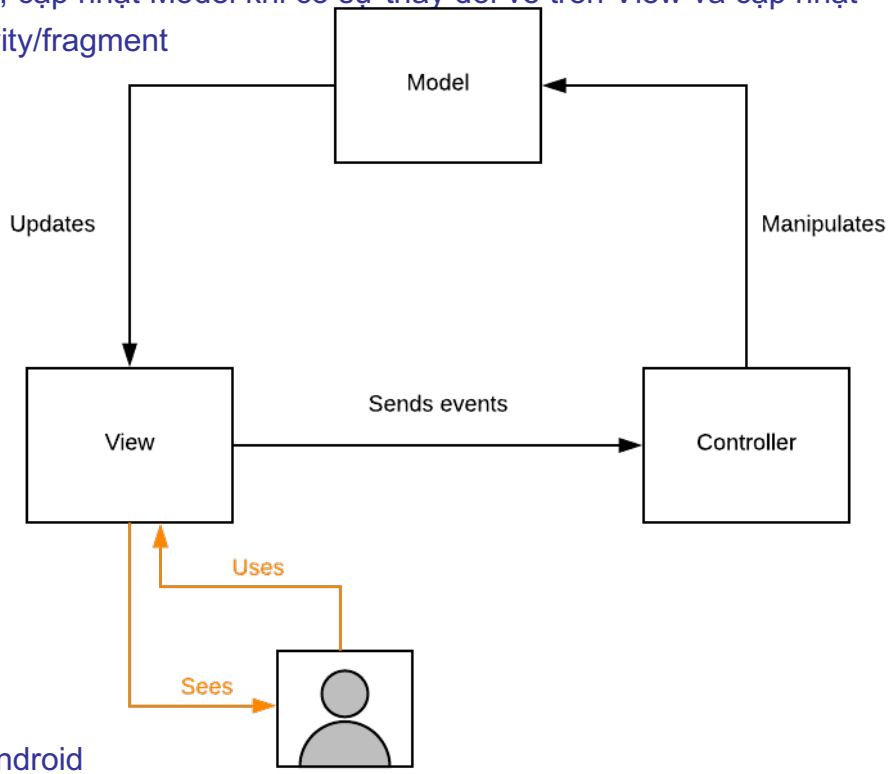
Nội dung

1. Mô hình Model – View - Controller (MVC)
2. Mô hình Model – View – Presenter (MVP)
3. Áp dụng mô hình MVP trong lập trình ứng dụng Android

Mô hình Model – View – Controller (MVC)

- Mô hình **MVC** thường được sử dụng để minh hoạ trong các tài liệu về Android, gồm 3 thành phần:
  - **Model**: chứa các dữ liệu (để trình bày/hiển thị), thường được lấy về từ network hay csdl → thường được xây dựng thành các class nhỏ, đơn giản để các thành phần khác có thể sử dụng
  - **View**: là những gì hiển thị với người dùng, đồng thời xử lý các tương tác từ người dùng lên màn hình (click, long click…). Chỉ nên chịu trách nhiệm hiển thị dữ liệu, không nên chứa các logic nghiệp vụ. Code nằm ở activity/fragment
  - **Controller**: là trung gian kết nối giữa View và Model, cập nhật Model khi có sự thay đổi về trên View và cập nhật View khi có sự thay đổi về Model. Code cũng ở activity/fragment
- Trong mô hình MVC, truyền thông giữa các thành phần như sau:
  - **View** trình bày/hiển thị trạng thái của **Model**
  - **Controller** xử lý các input từ **View**
  - **Model** lưu các kết quả đến từ **Controller**
- **MVC** khó để kiểm thử do các logic nghiệp vụ chồng chéo giữa các thành phần

→ Hầu hết code tập trung tại Activity/Fragment

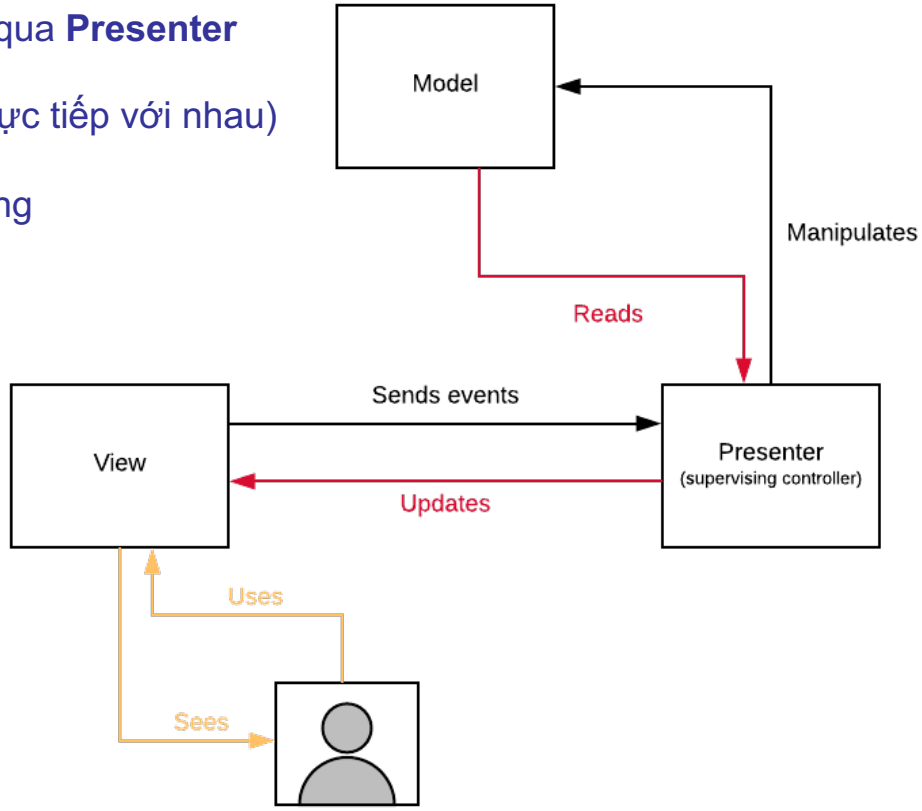


Mô hình Model – View - Presenter

- Mô hình **Model – View – Presenter** (**MVP**) phân tách **data model** khỏi **view** thông qua **presenter**
- Tập trung logic nghiệp vụ về **Presenter**
- **Presenter** còn chịu trách nhiệm điều chỉnh dữ liệu của **Model** để hiển thị lên **View**
- **Model** và **View** chỉ giao tiếp với nhau thông qua **Presenter**

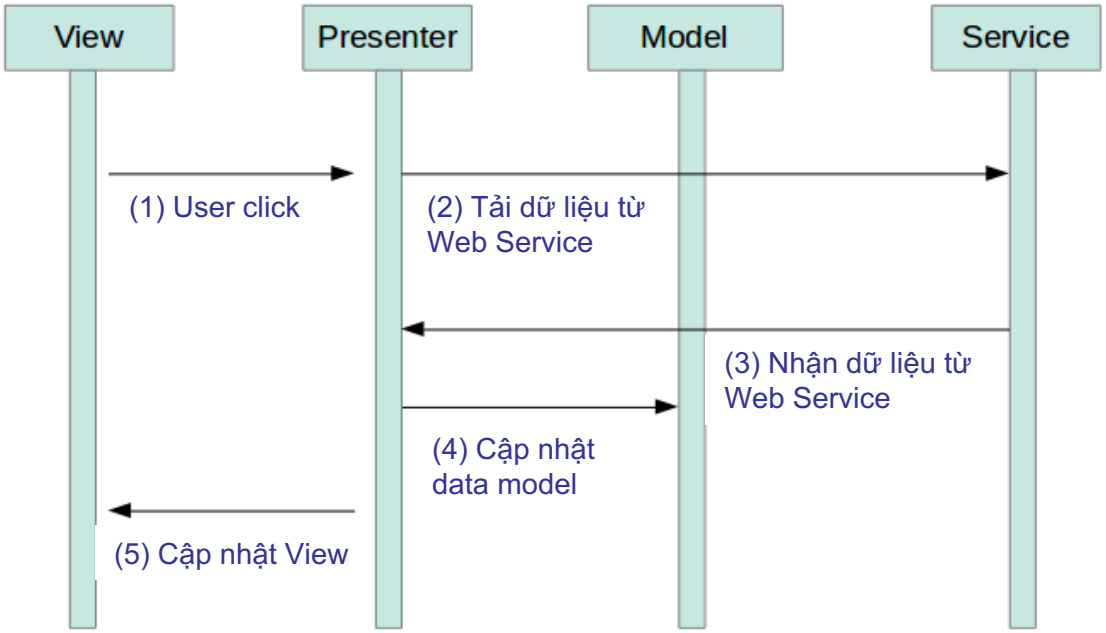
(khác với MVC, Model và View có thể giao tiếp trực tiếp với nhau)

- Giúp tăng cường khả năng kiểm thử ứng dụng



Mô hình Model – View – Presenter (2)

- Sơ đồ thứ tự sau minh hoạ một luồng các xử lý theo mô hình **MVP**



Các thành phần của mô hình MVP

- **View**
  - Giúp hiển thị nội dung ứng dụng, chỉ chứa các thành phần UI
  - Không chứa bất kỳ logic về cách dữ liệu được hiển thị như thế nào (hạn chế so với View của MVC)
  - Chứa các interface cho phép Presenter thao tác/chỉnh sửa nội dung của View
- **Presenter**
  - Xử lý input của người dùng, cập nhật view, thực thi các logic nghiệp vụ (thay thế cho Controller của MVC)
  - Tương tác với model, nạp dữ liệu từ model để cập nhật view
  - Presenter không nên phụ thuộc vào Android SDK
- **Model**
  - Mô tả dữ liệu (không thay đổi so với MVC)
  - Cho phép đọc và cập nhật dữ liệu
  - Thường thao tác với CSDL hoặc giao tiếp với Web Service

Khi nào sử dụng mô hình MVP

- Mô hình MVP giúp thực hiện các kiểm thử trên logic của presenter và giảm sự phụ thuộc vào Android SDK
- Tuy nhiên, sử dụng MVP thì nhà phát triển phải lập trình nhiều hơn (source code nhiều hơn). Các template Android chuẩn hiện tại cũng không sử dụng mô hình MVP.

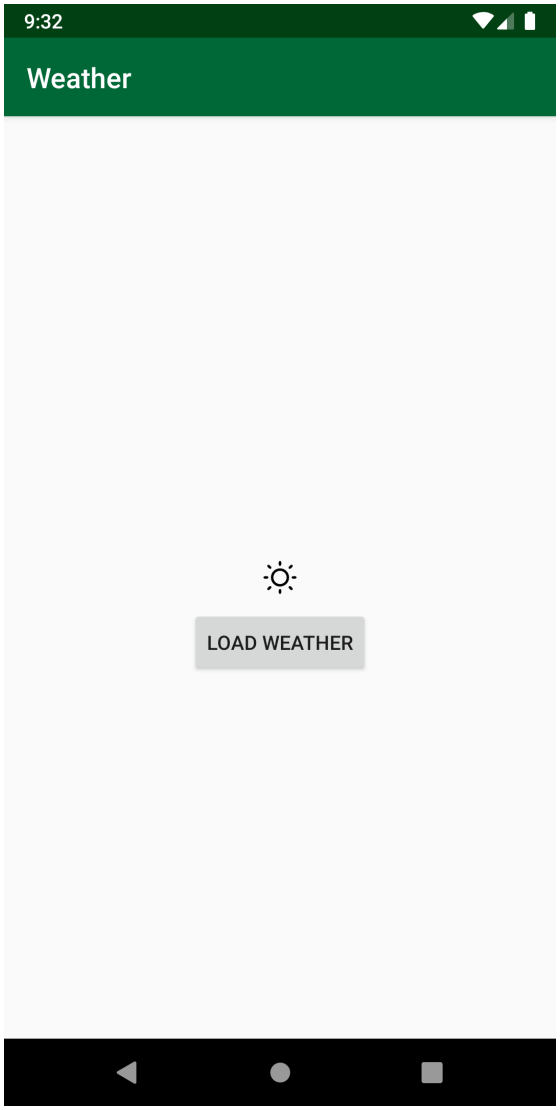
→ Khi cần thực hiện các **unit tests**, nên sử dụng mô hình MVP

Minh hoạ sử dụng mô hình MVP

- Ứng dụng minh hoạ hiển thị thời tiết hiện tại (weather), ứng với 2 sự kiện chính:
  - Khi ứng dụng được mở
  - Khi nhấn button “Load Weather”
- Để đơn giản, ứng dụng chỉ phát sinh ngẫu nhiên

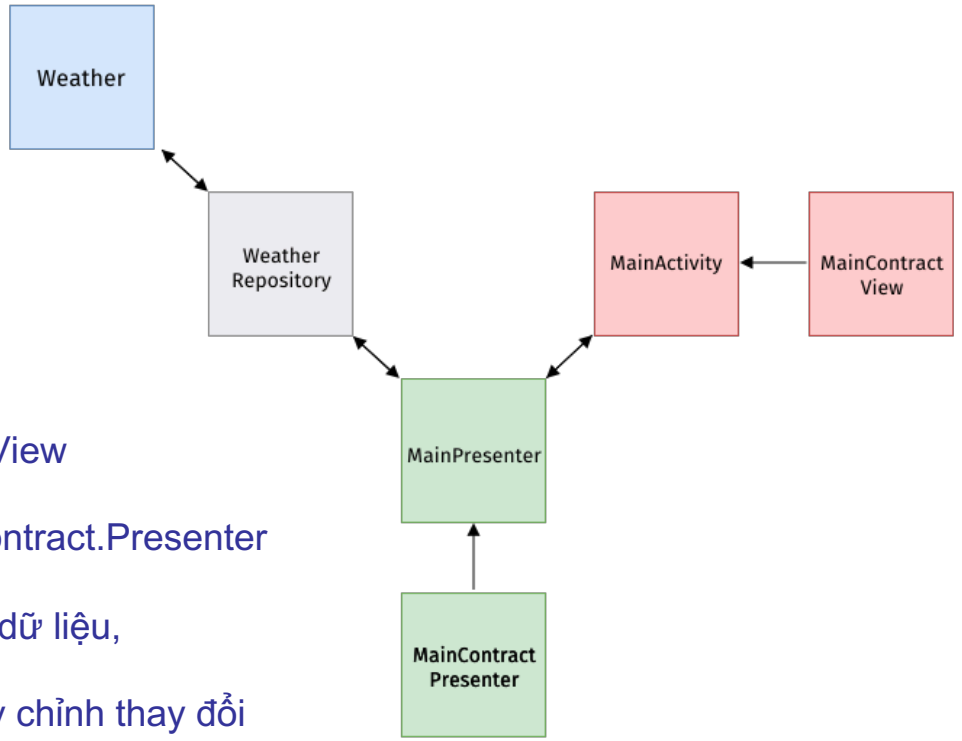
2 trường hợp thời tiết

- Sunny
- Raining



Mô hình ứng dụng

- Project được thiết kế theo mô hình **MVP** gồm các thành phần sau, các mũi tên chỉ ra cách các thành phần tương tác với nhau như thế nào
- Ứng dụng gồm một màn hình chính (**MainActivity**), do đó các thành phần tương ứng với nó sẽ có tiếp đầu ngữ "**Main**"
- **Model**: Weather
- **View**: MainActivity, kết thừa từ MainContract.View
- **Presenter**: MainPresenter, kế thừa từ MainContract.Presenter
- Thành phần WeatherRepository giúp truy cập dữ liệu, được tách rồi nhằm mục đích module hoá, giúp tuỳ chỉnh thay đổi lấy dữ liệu từ nguồn này sang nguồn khác một cách dễ dàng



Các bước xây dựng ứng dụng

- Tạo interface **BasePresenter**, đây là hình mẫu cho bất kỳ **Presenter** nào trong project cần implement theo. Trong ví dụ này, **BasePresenter** gồm phương thức onDestroy() tương ứng với lifecycle của activity

```
public interface BasePresenter {
    void onDestroy();
}
```

- Tạo interface **BaseView**, cũng là hình mẫu cho tất cả các View trong ứng dụng implement theo. Do mỗi View đều tương tác với một Presenter, nên mỗi View được cung cấp một kiểu generic T ứng với Presenter, và phải có phương thức setPresenter()

```
public interface BaseView<T> {
    void setPresenter(T presenter);
}
```

Các bước xây dựng ứng dụng (2)

- Tạo class **MainContract**, định nghĩa các interface cho **View** và **Presenter** ứng với màn hình **MainActivity**. Các interface này chỉ tạo ra cho một activity cụ thể (**MainActivity**).

```
public class MainContract {
    interface Presenter extends BasePresenter {
        void onViewCreated();
        void onLoadWeatherTapped();
    }

    interface View extends BaseView<Presenter> {
        void displayWeatherState(int weatherStateResourceId);
    }
}
```

- **MainContract.Presenter** được "callback" bởi **MainContract.View** khi xảy ra các sự kiện trên View:
  - onViewCreated() khi view được tạo
  - onLoadWeatherTapped() khi button "Load Weather" được nhấn
- **MainContract.View** được gọi để hiển thị thông tin thời tiết thông qua phương thức displayWeatherState()

Định nghĩa Presenter

- Tạo class **MainPresenter** kế thừa từ **MainContract.Presenter**, gồm 2 thành phần:
  - Một tham chiếu đến **MainContract.View** (lưu ý, chỉ tương tác với interface)
  - Một tham chiếu đến **WeatherRepository**, cũng là interface, giúp truy cập dữ liệu/model

```
public class MainPresenter implements MainContract.Presenter {
    MainContract.View view;
    WeatherRepository weatherRepository;

    public MainPresenter(MainContract.View view, WeatherRepository weatherRepository) {
        this.view = view;
        this.weatherRepository = weatherRepository;
    }
}
```

Định nghĩa Presenter (2)

- Tạo các phương thức private sau, bên trong class **MainPresenter**:

```
private void loadWeather() {
    Weather weather = weatherRepository.loadWeather();
    int resId = weatherResourceIdForWeatherState(weather.weatherState);
    view.displayWeatherState(resId);
}

private int weatherResourceIdForWeatherState(String weatherState) {
    if ("rain".equals(weatherState))
        return R.drawable.ic_umbrella;
    return R.drawable.ic_sun;
}
```

- **WeatherRepository** chịu trách nhiệm tải về thông tin thời tiết (từ network, database…). Ứng với thông tin thời tiết, tìm ra hình ảnh phù hợp để hiển thị.
- Thông qua tham chiếu view, gọi phương thức cập nhật hiển thị lên view: displayWeatherState()

Định nghĩa Presenter (3)

- Cuối cùng, override các phương thức sau:

```
@Override
public void onViewCreated() {
    loadWeather();
}

@Override
public void onLoadWeatherTapped() {
    loadWeather();
}

@Override
public void onDestroy() {
    this.view = null;
}
```

- Thực hiện "dọn dẹp" trong onDestroy() và thực hiện tải về thông tin thời tiết ứng với các sự kiện onViewCreated() và onLoadWeatherTapped()

Định nghĩa View

- Xây dựng **MainActivity** như sau:

```
public class MainActivity extends AppCompatActivity implements MainContract.View { // 1
    ImageView ivWeatherState;
    Button btnLoadWeather;

    MainContract.Presenter presenter; // 2

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        ivWeatherState = findViewById(R.id.ivWeatherState);
        btnLoadWeather = findViewById(R.id.btnLoadWeather);

        setPresenter(new MainPresenter(this, new RandomWeatherRepository())); // 3
        presenter.onViewCreated(); // 4

        btnLoadWeather.setOnClickListener(new android.view.View.OnClickListener() {
            @Override
            public void onClick(android.view.View view) {
                presenter.onLoadWeatherTapped(); // 5
            }
        });
    }
// …
}
```

Định nghĩa View (2)

- Tiếp tục **override** các phương thức từ interface MainContract.View, BaseView và Activity:

```
public class MainActivity extends AppCompatActivity implements MainContract.View {
    // …

    @Override
    public void displayWeatherState(int weatherStateResourceId) { // 6
        ivWeatherState.setImageDrawable(getResources().getDrawable(weatherStateResourceId,
getApplicationContext().getTheme()));
    }

    @Override
    public void setPresenter(MainContract.Presenter presenter) { // 7
        this.presenter = presenter;
    }

    @Override
    protected void onDestroy() { // 8
        presenter.onDestroy();
        super.onDestroy();
    }
}
```

Định nghĩa View (3)

- // 1 – Cho **MainActivity** kế thừa từ interface MainContract.View → MainActivity hoàn toàn là một View, không còn có vai trò là Controller như mô hình MVC
- // 2 – Thêm thuộc tính presenter, thay vì model weatherRepository → MainActivity không còn quản lý model, mà sẽ do presenter quản lý
- // 3 – Tạo ra đối tượng presenter và lưu lại giá trị này bằng phương thức setPresenter(). Lưu ý, tạo ra đối tượng **WeatherRepository** và đưa nó trở thành tham số đầu vào để tạo ra đối tượng presenter
- // 4 – Khi đã có đối tượng presenter, thực hiện callback onViewCreated() để tải về và trình bày dữ liệu thời tiết ứng với sự kiện mở màn hình MainActivity
- // 5 – Xử lý sự kiện click lên button "Load Weather" và thực hiện callback tới presenter → onLoadWeatherTapped()
- // 6 – override phương thức displayWeatherState() từ interface **MainContract.View** để hiển thị thông tin thời tiết, là một phần của view interface, được gọi từ presenter
- // 7 – override phương thức setPresenter() từ interface **BaseView**, để gán giá trị presenter
- // 8 – override phương thức onDestroy() từ class **Activity**, trong phương thức này thông báo đến presenter khi view bị huỷ. Presenter lợi dụng cơ hội này để "dọn dẹp" tất cả những gì không còn cần thiết cho tới thời điểm này

Định nghĩa Model

- Class **Weather** biểu diễn thông tin thời tiết có thể được xây dựng như sau:

```
public class Weather {
    String weatherState; // rain, cloud

    public Weather(String weatherState) {
        this.weatherState = weatherState;
    }
}
```

- Interface **WeatherRepository** đưa ra hình mẫu cho các thao tác trên model Weather mà các class kế thừa từ nó cần tuân theo:

```
public interface WeatherRepository {
    Weather loadWeather();
}
```

- Trong minh hoạ này, **RandomWeatherRepository** chịu trách nhiệm loadWeather() một cách ngẫu nhiên giúp đơn giản hoá cho minh hoạ. Có thể tạo các class khác để loadWeather() từ các Web Service (ví dụ: WebServiceWeatherRepository) cũng kế thừa từ interface WeatherRepository.

```
public class RandomWeatherRepository implements WeatherRepository {
    @Override
    public Weather loadWeather() {
        String[] weathers = {"rain", "cloud"};
        int index = (new Random()).nextInt(1000) % 2;
        Weather weather = new Weather(weathers[index]);
        return weather;
    }
}
```

Q&A