

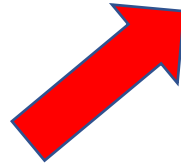
Overview


- We are going to consume an API from OpenData
 - https://opendata.vancouver.ca/api/records/1.0/search/?dataset=dog-off-leash-parks&q=&facet=geo_local_area
 - Dog off leash parks in Vancouver
- Our goal is too fetch this data asynchronously and display it in a *RecyclerView*, similar to recent lecture
- The biggest difference here is our code is going to follow the MVP (model-view-presenter) design architecture
 - Will be discussed throughout instruction

<https://opendata.vancouver.ca/pages/home/>



Theme		
	Business and economy	5
+	Culture and education	5
	Demographics	5
	Food and housing	6
	Geography and imagery	17
	Government and finance	32
	Parks, recreation, and pets	14
	Property and development	29
	Safety	4
	Streets and transportation	46
	Sustainability	2
	Water and sewer	14
	> Less	





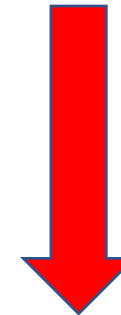
Dog off leash parks

The dog off-leash parks dataset is one of several that describe Public places.

Last modified September 27, 2021 3:04 AM

[public place](#)

[Table](#)
[Map](#)
[Analyze](#)
[Export](#)
[API](#)



and records.

[Submit](#)

https://api.records/1.0/search/?dataset=dog-off-leash-parks&q=&facet=geo_local_area

Creating our POJO

- Since the JSON object is more complex than the cartoon JSON from last lecture we are going to use an online tool to generate our POJO (plain old java object) class
 - <https://json2csharp.com/json-to-pojo>
- Create a new java class (*OffLeashParks.java*) and copy/paste the generated code
- It should look like something like the image on the next slide

```

public class OffLeashParks {

    public class Parameters{
        public String dataset;
        public String timezone;
        public int rows;
        public int start;
        public String format;
        public List<String> facet;
    }

    public class Geom{
        public String type;
        public List<List<List<Double>>> coordinates;
    }

    public class Fields{
        public String url;
        public String address;
        public Geom geom;
        public String geo_local_area;
        public String name;
    }

    public class Record{
        public String datasetid;
        public String recordid;
        public Fields fields;
        public Date record_timestamp;
    }

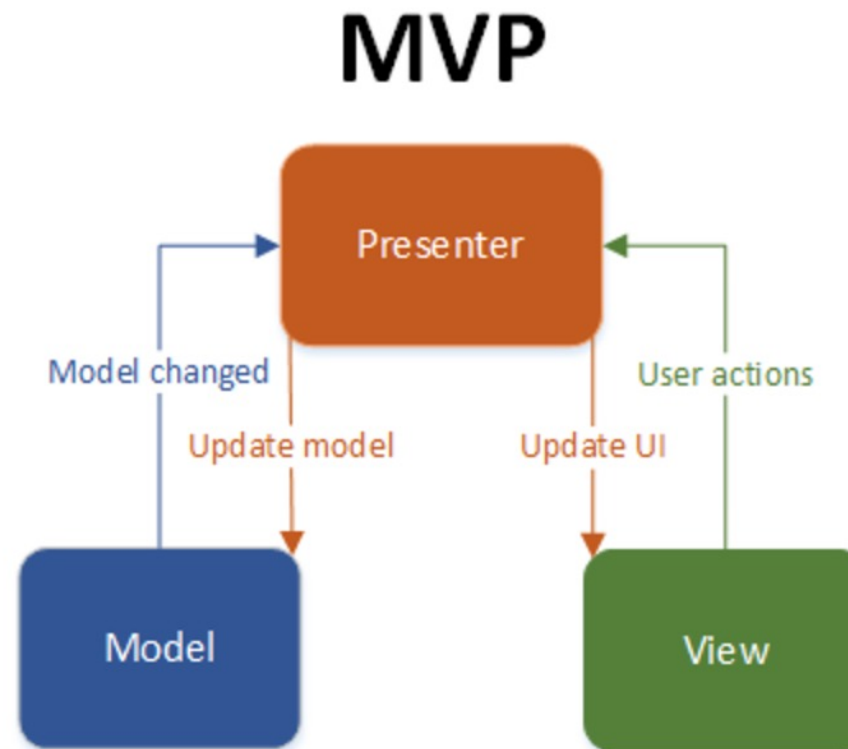
    public class Facet{
        public int count;
        public String path;
        public String state;
        public String name;
    }

    public class FacetGroup{

```

Model View Presenter

- MPV architecture is a common way we can set up our code so it is modular and scalable
 - model-view-presenter



Model View Presenter (cont.)

- In Android you can think of the MVP modules like this
 - Model
 - Represents and manages the data
 - View
 - Activities/Fragments
 - Views/layouts
 - Presenter
 - Mediator for the Model and View
 - No knowledge of View it is trying to update
 - Decides what happens when you interact with the View

Model View Presenter (cont.)

- Note that MVP is different then MVC (model-view-controller)
 - MVC vs MVP main differences
 - Your modules are more tightly coupled with MVC
 - Presenter/Controller has no relation to the View in MVP
 - View's are updated through an interface in MVP
 - To find some more info
 - Check out *Resources->Web Resources/Links* on LearningHub

Model View Presenter (cont.)

- Create a *MainPresenter.java* class and a *MainModel.java*
- To keep things simple lets think of the MVP as
 - Model: *MainModel.java*
 - View: *MainActivity*
 - Presenter: *MainPresenter.java*

MainPresenter

- Inside you *MainPresenter.java*, create an interface called *View* with one function
- Pass a *View* and *MainModel* through the constructor as well

```
public class MainPresenter {  
  
    public interface View{  
        void onUpdateRecycler(List<OffLeashParks.Record> areas);  
    }  
  
    private final MainModel mainModel;  
    private final View view;  
  
    public MainPresenter(View view, MainModel mainModel) {  
        this.mainModel = mainModel;  
        this.view = view;  
    }  
}
```

MainPresenter (cont.)

- Create a function inside *MainPresenter* like so

```
public void getDataFromModel() {  
  
    //List<OffLeashParks.Record> areas = mainModel.getData();  
    //view.onUpdateRecycler(areas);  
}
```

- You can see this function will get the data from the model and then *onUpdateRecycler* will be called here

MainActivity

- In your MainActivity.java, implement *MainPresenter.View* and override *onDataSuccess*

```
public class MainActivity extends AppCompatActivity implements MainPresenter.View {
```

```
    @Override  
    public void onUpdateRecycler(List<OffLeashParks.Record> areas) {  
  
    }  
}
```

MainActivity (cont.)

- As soon as our app runs we want to display a *RecyclerView* with the data from our API
- Lets fill out our *onCreate* in *MainActivity.java*
 - Here we should create our *MainPresenter* and *MainModel* objects
 - Get a reference to our *RecyclerView* and set the *LayoutManager* for it
 - Doesn't need adapter yet, because we don't have data for it
 - Our *MainPresenter* object should call the *getDataFromModel* method
- Hopefully you can see now that the Presenter (*MainPresenter*) has no knowledge of the *View* (*MainActivity*)
 - Take some time to absorb this, but it's all in the *View* interface

MainModel

- The *MainModel* will be fetching our data from the API and parsing it
- The Model represents and manages our data
 - *Persistence and business logic*
- Example Responsibilities
 - URL/DB/API management, get/post data, parsing
 - Note that all these operations can be done in separate classes, but should only interact with the *Model*, not with the *Presenter* or *View*

Parsing our JSON

- Explore the generated POJO and to see how you can retrieve the required data
- Hint: The whole JSON can be parsed into the *Root* class of the generated POJO
 - This is assuming you used the online tool I provided, other tools may create different structured POJO's
- We only want to display
 - Name of park
 - Local geo area
 - Address

```
public class Fields{  
    public String url;  
    public String address;  
    public Geom geom;  
    public String geo_local_area;  
    public String name;  
}
```