CMU 04-801: Advanced Mobile Application Development
Week 3: Local Data

Keeping data externalized is better than hard coding data into your app. Data is often incorporated into your app from a variety of sources in different formats. Outside of SQL databases the most common data formats are XML and JSON.

XML
XML stands for eXtensible Markup Language
XML is a markup language designed to structure data

- it looks a lot like HTML, the markup language that describes the structure of a web page
- both are derived from SGML (Standard Generalized Markup Language) (W3C)

XML was designed to describe data and to focus on what the data is

- defines tags and the structural relationships between them.
- XML tags are not predefined -- you must define your own tags.

It's up to the application using XML data to process the document.
Android has multiple parsers that handle XML

- **XMLPullParser** is the most widely used for XML parsing
- **SimpleXML** is a Java library that makes parsing XML a little less painful

JSON
JSON is a standard text-based format for representing structured data based on JavaScript object syntax. Although it was based on JavaScript it's easy to read (parse) and generate JSON in many programming environments.
JSON is light weight, structured and is an independent data exchange format that is used to parse data.
JSON uses name/value pairs to create objects of data.
Although XML is still used in many applications, JSON is now a more popular way to structure data.
JSON is considered simpler, less verbose, and faster than XML.
Most APIs make their data available in JSON format
Android provides classes to manipulate JSON data

- **JSONObject** https://developer.android.com/reference/org/json/JSONObject
  - Represents a JSON object as a modifiable set of name/value mappings.
    - Names are unique, non-null strings
    - Values may be any mix of `JSONObject`, `JSONArray`, Strings, Booleans, Integers, Longs, Doubles or `NULL`.
  - Many different get methods to access different types of data
    - getJSONObject() returns a JSON object
      - { } represents a JSON object
    - getJSONArray() returns an array
      - [ ] represents a JSON array
- **JSONArray** https://developer.android.com/reference/org/json/JSONArray
  - Represents a JSON array which can be the value for a given key
    - Values may be any mix of `JSONObject`, other `JSONArray`, Strings, Booleans, Integers, Longs, Doubles, or null
  - Many different get methods to access different types of data
- **JSONStringer**
  - Above classes and methods are used instead of JSONStringer
- **JSONTokener**

    - Used to parse a JSON string into a JSON object
- **JSONException**
    - handles all the exceptions related to JSON parsing.

There are manu JSON parsing libraries as well:

- Google's Gson library for converting between JSON and Java objects. Gson will also automatically map JSON keys into your model class. This helps to avoid needing to write boilerplate code to parse JSON responses yourself.
  https://github.com/google/gson/blob/master/UserGuide.md
- Jackson is a suite of data-processing tools for Java including the JSON parser/generator library, and matching data-binding library (POJOs to and from JSON)
  https://github.com/FasterXML/jackson
- Moshi is another JSON library for Android that makes it easy to parse JSON into Java objects and serialize Java objects as JSON. Moshi also includes Kotlin support.
  https://github.com/square/moshi

XML/JSON example: https://www.geeksforgeeks.org/difference-between-json-and-xml/

Resources

https://developer.android.com/guide/topics/resources/providing-resources

We've already used many different types of resources in Android Studio such as values, layouts, and drawables.

This is also where you would include any static data files that your app will use.

By externalizing your app resources, you can access them using resource IDs that are generated in the project's `R` class.

There are many directories supported in your resources directory including the following for data:

- raw – used for files in their raw form including JSON.
    - To open these resources with a raw `InputStream`, call `Resources.openRawResource()` with the resource ID, which is `R.raw.`*`filename`*.
    - InputStream is an abstract class that you can use to manipulate the bytes in the JSON file
    - The Scanner class makes it even easier to parse primitive types and strings including JSON files using regular expressions.
      https://developer.android.com/reference/java/util/Scanner
        - Scanner breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various `next` methods.
        - The delimiter pattern `\A` means 'the beginning of the input'
        - `.next()` reads the next token
        - Since a JSON file is really one JSON string we only need one line to grab the whole JSON file

```
String jsonString = new Scanner(inputStream).useDelimiter("\\A").next();
```

- assets – used for files if you need access to original file names and file hierarchy
    - files in `assets/` aren't given a resource ID, so you can read them only using `AssetManager`.
- xml – used for XML files
    - can be read at runtime by calling getResources().getXml(R.xml.*filename*)
        - returns a XMLResourceParser which is a subclass of XMLPullParser
        - more efficient than adding it as an asset and opening it as an input stream

Harry Potter

Create a new project called HarryPotter
Empty Activity template
Package name: the fully qualified name for the project
Minimum SDK: API 21 (21 is the minimum API for Material Design)
Language: Java

Add Material Components into the app's grade file

```
implementation 'com.google.android.material:material:1.1.0-rc01'
```

Add colors, create a theme with a Material Components parent theme, and apply it to your app.

Layout

Add any images you'll be using in your layout such as hogwarts.png into the drawable folder.
In activity _main.xml
Remove the textview and add an ImageView to the top of the view and chose the hogwarts image.

```
android:src="@drawable/hogwarts"
```

Add any missing constraints. If you want it with no gap/border add

```
android:adjustViewBounds="true"
```

Having a content description for an image is optional but makes your app more accessible.
In your strings.xml add

```
<string name="description">Header image</string>
```

In activity_main.xml use it for the contentDescription

```
android:contentDescription="@string/description"
```

Make sure Autoconnect(magnet) is turned on.

In the layout file in design view drag out a RecyclerView to display the list(can be found in common or containers) below the image.
Add Project Dependency to add the support library with the RecyclerView. This will add it to your build.gradle(module: app) file.

```
implementation 'androidx.recyclerview:recyclerview:1.1.0'
```

Give the RecyclerView an id `android:id="@+id/recyclerView"`
Add start, end, top, and bottom constraints of 0.
Make sure the RecyclerView has layout_width and layout_height set to "match_constraint" (0dp).

List Item Layout

We also need a layout file to determine what each row in our list will look like. Ours will be super simple with just a TextView.
File | New | Layout resource file
File name: list_item
Root element: android.support.constraint.ConstraintLayout

```
androidx.constraintlayout.widget.ConstraintLayout
```

Source set: main
Directory name: layout

Change the constraint layout's height to "wrap_content". If the height is match_constraint each text view will have the height of a whole screen.

```
android:layout_height="wrap_content"
```

Drag out a TextView and make sure it has an id (textView)
Make sure the TextView has layout_width and layout_height set to "wrap_content".
I also added some top and bottom padding so the text is in the vertical center of the row, some start padding, and made the text larger.

```
android:textAppearance="@style/TextAppearance.MaterialComponents.Headline6"
android:paddingBottom="5dp"
android:paddingTop="5dp"
android:paddingStart="10dp"
```

Once you don't need the default text to help with layout, remove it.

Java class

We're going to create a custom Java class to represent the data we'll be using.
First create a new package for our model. In the java folder select the tulips folder (not androidTest or test)
File | New | Package
Name: model

Then select the new model package and add a new class called Bulb.
File | New | Java class
Name: Character
Kind: Class

The Character class will store the character name and a String for the URL (info).
We'll have a constructor that takes name and info as arguments. We'll also need getter and setter methods for the data members.
For the constructor, getter, and setter methods Android Studio can generate the code for you.
Code | Generate | Constructor (pick whichever data members you want as arguments, we want both)
Code | Getter and Setter

```
public class Character {
    private String name;
    private String picture;
    private String info;

    public Character(String name, String info) {
        this.name = name;
        this.info = info;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public String getInfo() {
        return info;
    }

    public void setInfo(String info) {
        this.info = info;
    }
}
```

Now create another package called sample and then a Java class in sample called JSONData where we will load and parse our JSON data.
File | New | Package
Name: sample

Then select the new model package and add a new class called Bulb.
File | New | Java class
Name: JSONData
Kind: Class

```
public class JSONData {
    public static List<Character> characterList;

    static {
        characterList = new ArrayList<>();
    }

    public static List<Character> getJSON(Context context){
        String json = null;
        json = loadJSONFromRes(context); //load JSON from resource
        characterList = parseJSON(json); //parse JSON
        return characterList;
    }

    private static String loadJSONFromRes(Context context) {
       //opens the raw JSON file and assigns it to an InputStream instance
       InputStream inputStream =
context.getResources().openRawResource(R.raw.harrypotter);

       //stores the JSON as a String
       String jsonString = new
Scanner(inputStream).useDelimiter("\\A").next();
        return jsonString;
    }

     private static List<Character> parseJSON(String jsonString){
         if (jsonString != null) {
             try {
                 //create JSONObject
                 JSONObject jsonObject = new JSONObject(jsonString);

                 //create JSONArray with the value from the characters key
```

```
            JSONArray characterArray =
jsonObject.getJSONArray("characters");

            //loop through each object in the array
            for (int i =0; i < characterArray.length(); i++) {
                JSONObject charObject =
characterArray.getJSONObject(i);

                //get values for name and info keys
                String name = charObject.getString("name");
                String info = charObject.getString("info");

                //create new Character object
                Character character = new Character(name, info);

                //add character object to our ArrayList
                characterList.add(character);
            }
        } catch (JSONException e) {
            e.printStackTrace();
        }
    }
    return characterList;
}
```

In my sample app I include in my comments the code if you don't use the Scanner class.

Adapter

Now we need to create an adapter which will be used to populate the data into the RecyclerView. The adapter's role is to convert an object at a given position into a list row item to be inserted in the list.

In the java folder create a new class for our adapter.
File | New | Class
Name: MyListAdapter
Superclass: androidx.recyclerview.widget.RecyclerView.Adapter

With a RecyclerView the adapter requires the existence of a ViewHolder object which describes and provides access to all the views within each item row.
To indicate that our class will be implementing the ViewHolder class add it to the class definition.

```
public class MyListAdapter extends RecyclerView.Adapter<
MyListAdapter.ViewHolder>
```

You will give you errors until you implement the required methods.
Select the light bulb and choose implement methods to get rid of the error.
Every adapter has three required methods that we need to implement.
You will still have an error because we need to define the ViewHolder class which will set up the bindings to the view in the layout file.
In our MyListAdapter class we'll create a ViewHolder class to render the item and a default constructor.

```
public class MyListAdapter extends RecyclerView.Adapter<
MyListAdapter.ViewHolder> {
    class ViewHolder extends RecyclerView.ViewHolder {
        TextView mTextView;

     //constructor method
     public ViewHolder(@NonNull View itemView) {
         super(itemView);
         mTextView = itemView.findViewById(R.id.textView);
     }
}
```

This should have fixed the errors.
With the viewholder defined, let's set up the MyListAdapter class. We'll define a list of characters, Context and a constructor that takes both. We'll use this later in MainActivity to define an instance of this class.

```
private List<Character> mCharacterList;
private Context mContext;

public MyListAdapter (List<Character> mCharacterList, Context mContext) {
    this.mCharacterList = mCharacterList;
    this.mContext = mContext;
}
```

Now we'll implement the 3 required methods.
OnCreateViewHolder() is called automatically by the adapter each time it needs to display a data item. When you inflate the xml layout file, you get a view, and then you wrap that in an instance of your ViewHolder class and return that object.

```
@NonNull
@Override
public MyListAdapter.ViewHolder onCreateViewHolder(@NonNull ViewGroup
parent, int viewType) {
    LayoutInflater inflater = LayoutInflater.from(mContext);
    View itemView = inflater.inflate(R.layout.list_item, parent, false);
    ViewHolder viewHolder = new ViewHolder(itemView);
    return viewHolder;
}
```

OnBindViewHolder is called each time the adapter needs to display a new data item. It passes in the reference to the ViewHolder and the position of the data item in the collection. The job of OnBindViewHolder is to take that data object and display its values.

```
@Override
public void onBindViewHolder(@NonNull MyListAdapter.ViewHolder holder, int
position) {
    Character character = mCharacterList.get(position);
    holder.mTextView.setText(character.getName());
}
```

getItemCount() returns the number of items in the collection.

```
    @Override
    public int getItemCount() {
        return mCharacterList.size();
    }
```

Go into MainActivity.java.
First I'll get access to the array list of characters from my JSONData class to use in this class.

With the adapter set up, in onCreate() we need to instantiate an adapter with our sample data.
Then we need to set the adapter to the RecyclerView.

We also need to set a Layout Manager for our RecyclerView instance. A layout manager positions item views inside a RecyclerView and determines when to reuse item views that are no longer visible to the user.

We're using a LinearLayoutManager which shows the data in a simple list — vertically or horizontally (by default vertically).

```
public class MainActivity extends AppCompatActivity {
     List<Character> potterList=JSONData.characterList;

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

        //get data
        if (potterList.isEmpty()) {
            potterList = JSONData.getJSON(this);
        }

        //get the recycler view
        RecyclerView recyclerView = findViewById(R.id.recyclerView);

        //divider line between rows
        recyclerView.addItemDecoration(new DividerItemDecoration(this,
LinearLayoutManager.VERTICAL));

        //define an adapter
        MyListAdapter adapter = new MyListAdapter(potterList, this);

        //assign the adapter to the recycle view
        recyclerView.setAdapter(adapter);

        //set a layout manager on the recycler view
        recyclerView.setLayoutManager(new LinearLayoutManager(this));
    }
}
```

You should be able to run it and see your list of characters.

Item click listener

To attach a click handlers to items in our MyListAdapter class we'll create an interface with a method, define an instance of the interface, and pass it to the MyListAdapter constructor.

```
public interface ItemClickListener{
    void onItemClick(int position);
}

private ItemClickListener mItemClickListener;

public MyListAdapter(List<Character> mCharacterList, Context mContext,
ItemClickListener mItemClickListener) {
    this.mCharacterList = mCharacterList;
    this.mContext = mContext;
   this.mItemClickListener = mItemClickListener;
}
```

In the ViewHolder class we'll implement View.onClickListener and implement onClick so it calls the method in our interface with the correct list item position.

```
class ViewHolder extends RecyclerView.ViewHolder implements
View.OnClickListener{
    TextView mTextView;

    //constructor method
    public ViewHolder(@NonNull View itemView) {
        ….
        mTextView.setOnClickListener(this);
    }

    @Override
    public void onClick(View v) {
        mItemClickListener.onItemClick(getAdapterPosition());
    }
}
```

In MainActivity we'll implement the interface and override the interface's method to define what should happen when a row is clicked.

```
public class MainActivity extends AppCompatActivity implements
MyListAdapter.ItemClickListener {}

@Override
public void onItemClick(int position) {
    Intent intent = new Intent(MainActivity.this,
DetailActivity.class);
    intent.putExtra("id", position);
    startActivity(intent);
}
```

DetailActivity will give you an error because we haven't created it yet, we'll do that next.
We use the id to pass which character was selected.

When creating the adapter we also need to pass a third parameter now, "this" which refers to the MainActivity.

```
MyListAdapter adapter = new MyListAdapter(potterList, this, this);
```

Web Content

https://developer.android.com/guide/webapps/webview.html

There are two different ways to handle web content.

1. Open up a browser, using an implicit intent
2. Use an Android WebView to display web pages embedded in your app. WebView is an Android UI component that is an embedded browser that can be integrated to Android application
    a. WebView uses the WebKit rendering engine to display web pages and includes methods to navigate forward and backward through history, zoom, perform searches and more.

In DetailActivity we're going to use a WebView load the web page associated with the character.

DetailActivity

Now let's create the DetailActivity activity.
New | Activity | Empty
Activity name: DetailActivity
Check Generate Layout File
Package name: com.examples.tulips

Go into the activity_detail layout file and add delete the TextView.
Add a WebView (Widgets).
Give it the id webView and add constraints. I created constraints on all sides with a value of 0 and updated layout_width and layout_height to match_constraint(0dp).

In DetailActivity.java we'll get the data sent from the intent to populate the view.

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

    //get id from the intent
    int id = (Integer)getIntent().getExtras().get("id");
        Character character = JSONData.characterList.get(id);
        String webString = character.getInfo();

    //web view
    webView = findViewById(R.id.webView);
    webView.setWebViewClient(new WebViewClient(){
        @Override
        public boolean shouldOverrideUrlLoading(WebView view,
WebResourceRequest request) {
            //tells Android to use webview instead of opening in a browser
            return super.shouldOverrideUrlLoading(view, request);
        }
    });
    webView.getSettings().setJavaScriptEnabled(true);
    webView.loadUrl(webString);
}
```

The final step is to add internet permission to the AndroidManifest file. This will enable the WebView object to access the internet and download web pages. This must be a child node of the <manifest> tag.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Now when you run the app the detail view should show the Wikipedia page for whichever character you clicked on.
If you get an error for access denied and you have the above permission, uninstall the app from the device/emulator and run again.

If you navigate to other pages in the WebView you'll notice that the back button doesn't keep track of the web history and just takes you back to the previous activity. You can handle web page navigation by overriding the onKeyDown() method in DetailActivity.

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
    // Check if the key event was the Back button and if there's history
    if ((keyCode == KeyEvent.KEYCODE_BACK) && webView.canGoBack()) {
        webView.goBack();
        return true;
    }
    // If it wasn't the Back key or there's no web page history, bubble up
to the default
    // system behavior (probably exit the activity)
    return super.onKeyDown(keyCode, event);
}
```

To add up navigation go into the manifests/AndroidManifest.xml file and add a parent activity name for DetailActivity. This will automatically add the up navigation to navigate to the parent.

```
<activity android:name=".DetailActivity"
    android:parentActivityName=".MainActivity">
</activity>
```

If you didn't want to use a WebView to embed the web page, but instead just send the page to be opened in a browser you could create an implicit intent in onItemClick() in MainActivity instead of using DetailActivity. (sample app includes this in comments)

```
@Override
public void onItemClick(int position) {
    //opens web page in browser
    Intent intent = new Intent((Intent.ACTION_VIEW));
    Character character = JSONData.characterList.get(position);
    String webString = character.getInfo();
    intent.setData(Uri.parse(webString));

    //starts new activity
    startActivity(intent);
}
```