Local Data

Static Data

- POJO Model objects are best - simple properties
- Code generators work well with POJOs
- Getters/Setters, Constructor, toString
- Parcelable

Collections

- Collections…
- List - List<type>
  - ArrayList - concrete class implementing List - ArrayList<>()
  - .add, etc. functions
  - Ordered, Non-unique
  - Foreach loop
  - Collections.sort() or .sort(List, Comparator…) to compare custom

```
for (DataItem item : dataItemList) {
        tvOut.append(item.getItemName() + "\n");
    itemNames.add(item.getItemName());
}
```

```
Collections.sort(dataItemList, new Comparator<DataItem>() {
    @Override
    public int compare(DataItem o1, DataItem o2) {
        return o1.getItemName().compareTo(o2.getItemName());
    }
});
```

Collections

- Map - Map<keyType, valueType>
  - HashMap - concrete class implementing Map - HashMap<>()
  - .put, etc. functions

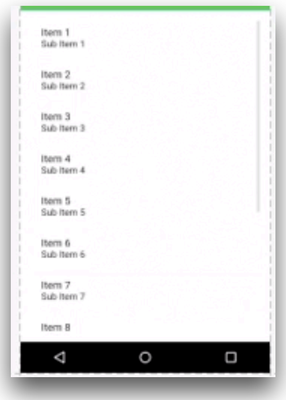
UI

- Various widgets to display data
- ListView - older
  - ArrayAdapter
- RecyclerView - newer

List View

- ListView - older
  - ArrayAdapter specifying type (e.g,. String)
  - List item layout (predefined or custom) - (Ctrl/Cmd click to view the layout)
  - Bind adapter to listview

```
<ListView
    android:id="@android:id/list"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```




```
ArrayAdapter<String> adapter = new ArrayAdapter<String>(
        this, android.R.layout.simple_list_item_1, itemNames);

ListView listView = (ListView) findViewById(android.R.id.list);
listView.setAdapter(adapter);
```

Custom ArrayAdapter

- Subclass ArrayAdapter
- Properties for the model (List) and inflater
- Constructor - call super and store model and inflater from context

```
public class DataItemAdapter extends ArrayAdapter<DataItem> {

    List<DataItem> mDataItems;
    LayoutInflater mInflater;

    public DataItemAdapter(Context context, List<DataItem> objects) {
        super(context, R.layout.list_item, objects);

        mDataItems = objects;
        mInflater = LayoutInflater.from(context);

    }
```

Custom ArrayAdapter

- Override getView function
- Create return View if necessary (null passed in)
- Get UI items and set items to display
- Create and bind instance to ListView (e.g., MainActivity)

```
@NonNull
@Override
public View getView(int position, View convertView, ViewGroup parent) {

    if (convertView == null) {
        convertView = mInflater.inflate(R.layout.list_item, parent, false);
    }

    TextView tvName = (TextView) convertView.findViewById(R.id.itemNameText);
    ImageView imageView = (ImageView) convertView.findViewById(R.id.imageView);

    DataItem item = mDataItems.get(position);

    tvName.setText(item.getItemName());
    imageView.setImageResource();

    return convertView;
}
```

@DrawableRes int resId

Custom ArrayAdapter

- Subclass ArrayAdapter
- Properties for the model (List) and inflater
- Constructor - call super and store model and inflater from context

```
public class DataItemAdapter extends ArrayAdapter<DataItem> {

    List<DataItem> mDataItems;
    LayoutInflater mInflater;

    public DataItemAdapter(Context context, List<DataItem> objects) {
        super(context, R.layout.list_item, objects);

        mDataItems = objects;
        mInflater = LayoutInflater.from(context);

    }
```

Images

- Loading an image from the Drawable resources

```
String imageFile = “something.jpg”
InputStream inputStream = mContext.getAssets().open(imageFile);
Drawable d = Drawable.createFromStream(inputStream, null);
```

RecyclerView

- Newer, smoother
- In own library, backwards compatible
- Needs dependency - same version as appcompat

```
implementation 'com.android.support:appcompat-v7:28.0.0'
implementation 'com.android.support:recyclerview-v7:28.0.0'
```

RecyclerView

- Define in UI:

```
<android.support.v7.widget.RecyclerView
    android:id="@+id/rvItems"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layoutManager="LinearLayoutManager"/>
```

- May need the app namespace if not added:

```
<RelativeLayout xmlns:android="http://schemas.android.co
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

RecyclerView

- Adapter extends RecyclerView.Adapter<type> - type will be defined later
- Properties for model and context
- Constructor to take items and store in properties

```
public class DataItemAdapter extends RecyclerView.Adapter<DataItemAdapter.ViewHolder> {

    private List<DataItem> mItems;
    private Context mContext;

    public DataItemAdapter(Context context, List<DataItem> items) {
        this.mContext = context;
        this.mItems = items;
    }
```

RecyclerView

- Override onCreateViewHolder taking ViewGroup and view type
- Inflate UI with layout, parent and ‘attach to root’ (parent passed in)
- Create ViewHolder instance with View and return it.
- If using a custom list_item, RecyclerView does better with wrap_content:

```
android:layout_width="match_parent"
android:layout_height="wrap_content"
```

```
@Override
public DataItemAdapter.ViewHolder onCreateViewHolder(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;
}
```

RecyclerView

- Override onBindViewHolder with ViewHolder and position
- Configure ViewHolder passed in

```
@Override
public void onBindViewHolder(DataItemAdapter.ViewHolder holder, int position) {
    DataItem item = mItems.get(position);

    try {
        holder.tvName.setText(item.getItemName());
        String imageFile = item.getImage();
        InputStream inputStream = mContext.getAssets().open(imageFile);
        Drawable d = Drawable.createFromStream(inputStream, null);
        holder.imageView.setImageDrawable(d);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

RecyclerView

- Override getItemCount

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

RecyclerView

- ViewHolder - extends RecyclerView.ViewHolder
- Properties for UI items
- Constructor takes View, store UI Items as necessary

```
public static class ViewHolder extends RecyclerView.ViewHolder {

    public TextView tvName;
    public ImageView imageView;
    public ViewHolder(View itemView) {
        super(itemView);

        tvName = (TextView) itemView.findViewById(R.id.itemNameText);
        imageView = (ImageView) itemView.findViewById(R.id.imageView);
    }
}
```

Local Data Lab 1

1. Create a new Android Studio project.

2. Add the recyclerview dependency

3. Resync the Gradle scripts.

4. In activity_main.xml change the layout_height to “wrap_content”

```
implementation 'com.android.support:recyclerview-v7:28.0.0'
```

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

Lab 1

1. Create a new Model object called DataItem.

2. Use Code Generators to create getters/setters and constructor.

3. In MainActivity create a property for the data:

4. In onCreate, create the instances of the DataItem and add them to the List.

```
class DataItem {
    String itemName;
    String image;
```

```
List<DataItem> dataItems;
```

```
dataItems = new ArrayList<>();
for (int i=0; i<100; i++) {
    DataItem di = new DataItem("Name" + i, "image" + i + ".jpg");
    dataItems.add(di);
}
```

Lab 1

1. Add a RecyclerView (remove TextView) to the activity_main.xml layout.

2. In MainActivity onCreate, get the RecyclerView from the UI. We need to set the RecyclerView adapter but we don't have one yet.

```
<android.support.v7.widget.RecyclerView
    app:layoutManager="android.support.v7.widget.LinearLayoutManager"
    android:id="@+id/recyclerView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

```
RecyclerView rv = findViewById(R.id.recyclerView);
```

Lab 1

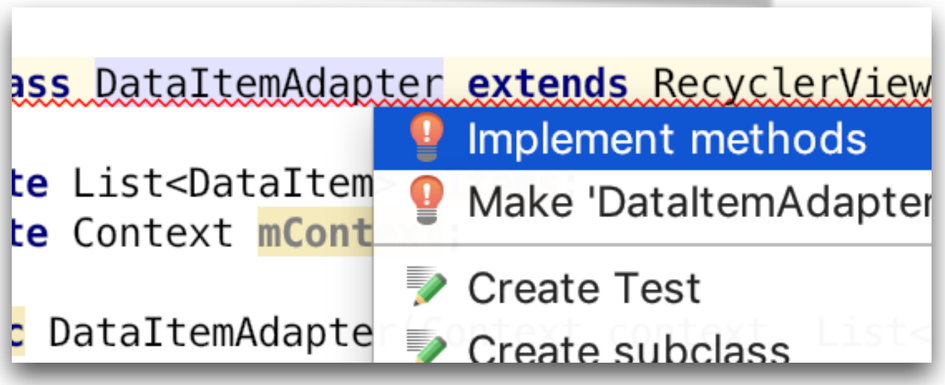
1. Create a new class (DataItemAdapter) that's a subclass of RecyclerView.Adapter that holds DataItemAdapter.ViewHolder items (not yet define).

2. Create properties for the Model items and the Context.

3. Use an Intent Action to generate the necessary Override methods for the class.

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

```
private List<DataItem> mItems;
private Context mContext;
```



Lab 1

1. Create a constructor to store the properties.

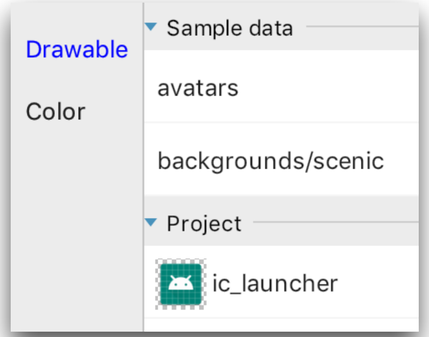
2. In the getItemCount function, return the size of the model property.

```
public DataItemAdapter(Context context, List<DataItem> items) {
    this.mContext = context;
    this.mItems = items;
}
```

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

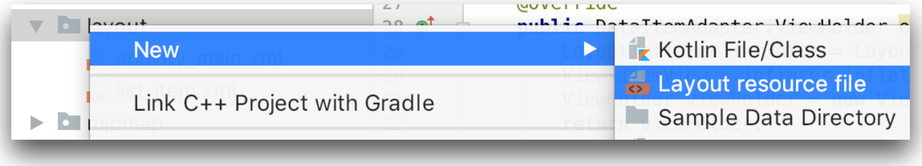
Lab 1

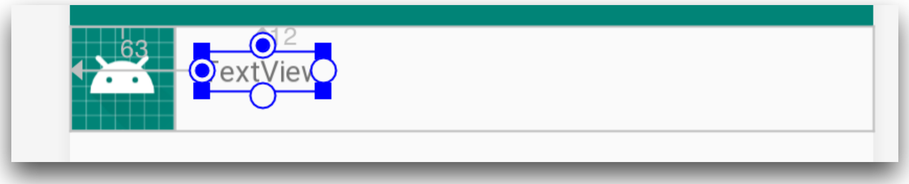
1. Under the res/layout folder, create a new layout resource file called list_item.xml



2. Define the UI to have an ImageView and TextView. When prompted for the image to display, use the Project icon.

3. Set and note the id's for these items.






```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="50dp"
    android:layout_height="50dp"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_centerVertical="true"
    app:srcCompat="@mipmap/ic_launcher" />

<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginStart="63dp"
    android:layout_marginTop="12dp"
    android:text="TextView" />
```

Lab 1

1. At the bottom, but inside the DataItemAdapter class, define the ViewHolder class.

2. Define properties for the TextView and ImageView.

3. In the constructor, set the properties to the UI items from the list item based on the id's used in the XML.

```
public static class ViewHolder extends RecyclerView.ViewHolder {

    public TextView tvName;
    public ImageView imageView;
    public ViewHolder(View itemView) {
        super(itemView);

        tvName = (TextView) itemView.findViewById(R.id.textView);
        imageView = (ImageView) itemView.findViewById(R.id.imageView);
    }
}
```

Lab 1

1. In DataItemAdapter, implement the onCreateViewHolder to inflate the UI, instantiate the ViewHolder and return it. (NOTE: param names may vary)

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

Lab 1

1. In DataItemAdapter, implement the onBindViewHolder to get the item by the position and set the UI items. (NOTE: param names may vary)

2. The image won't exist unless you added images to your project.

```
@Override
public void onBindViewHolder(DataItemAdapter.ViewHolder holder, int position) {
    DataItem item = mItems.get(position);

    try {
        holder.tvName.setText(item.getItemName());
        String imageFile = item.getImage();
        InputStream inputStream = mContext.getAssets().open(imageFile);
        Drawable d = Drawable.createFromStream(inputStream, null);
        holder.imageView.setImageDrawable(d);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

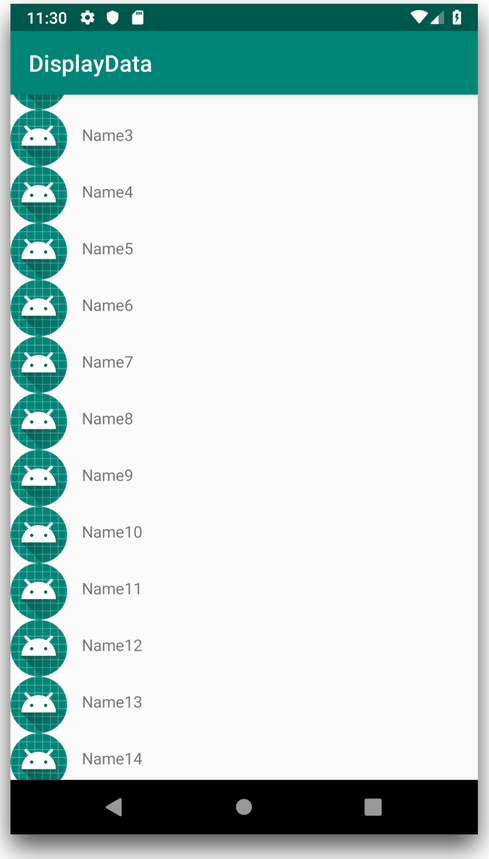
Lab 1

1. In MainActivity onCreate, create the DataItemAdapter instance.

2. Use the rv instance and set the adapter.

3. Run the app and test.

```
DataItemAdapter dia = new DataItemAdapter(this, dataItems);
rv.setAdapter(dia);
```



User Events

- Add View property
- Set in constructor

```
public static class ViewHolder extends RecyclerView.ViewHolder {

    public TextView tvName;
    public ImageView imageView;
    public View view;

    public ViewHolder(View itemView) {
        super(itemView);

        tvName = (TextView) itemView.findViewById(R.id.textView);
        imageView = (ImageView) itemView.findViewById(R.id.imageView);
        view = itemView;
    }
}
```

User Events

- Create OnClickListener in onBindViewHolder

```
viewHolder.view.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

    }
});
```

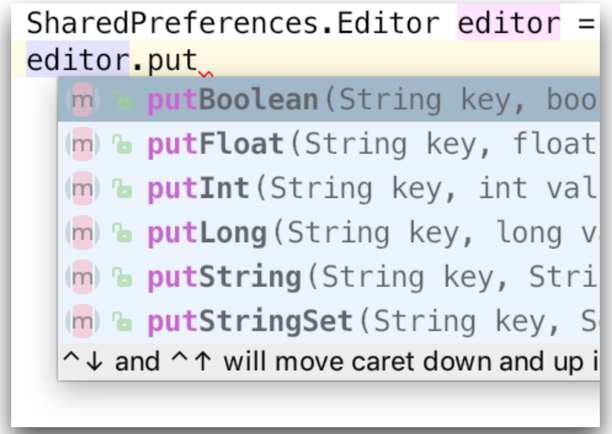
Shared Preferences

- Stored in XML files (unencrypted)
- Can have multiple preference sets
- Private to current app (internal storage)
- Default and named preference sets
- Key-Value pairs
  - Key is String
  - Value is primitive or String

Shared Preferences

```
SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(this);
SharedPreferences.Editor editorDef = preferences.edit();
editorDef.remove("addr");
editorDef.apply();
```

- Default SharedPreferences
- Or named
- Editor object: get/set functions



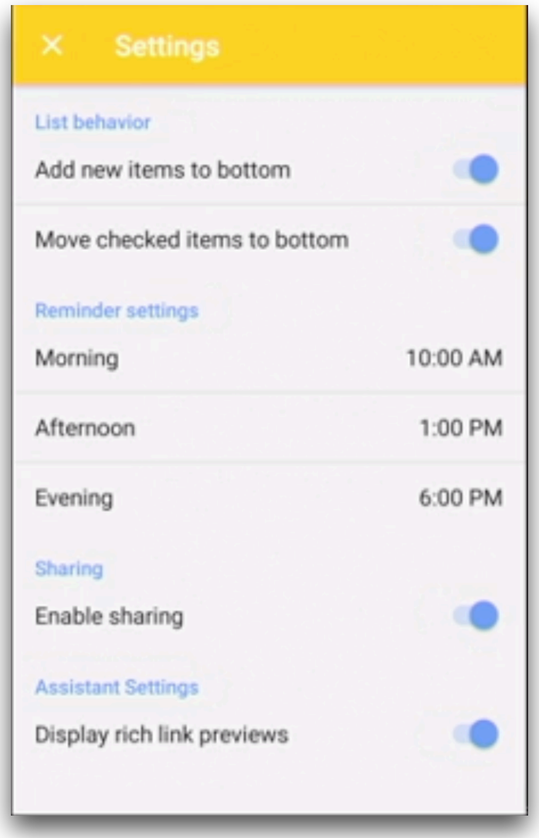

```
SharedPreferences.Editor editor = getSharedPreferences("MYPREFS", MODE_PRIVATE).edit();
editor.putString("username", "bearc");
editor.apply();
```

Shared Preferences

- Can be handed via a Preferences Activity
- New dependency
- Create new activity and basic layout, set id

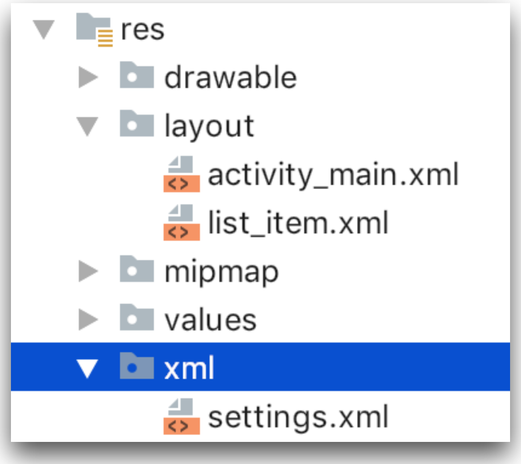
```
implementation 'com.android.support:preference-v7:28.0.0'
```

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    android:id="@+id/prefs_activity"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".PrefsActivity">

</FrameLayout>
```



Shared Preferences

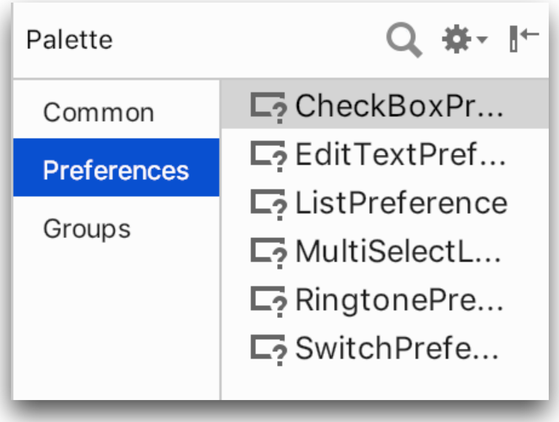
- New res > xml directory
- New XML Resource File: settings.xml
- Brings it up as a PreferenceScreen



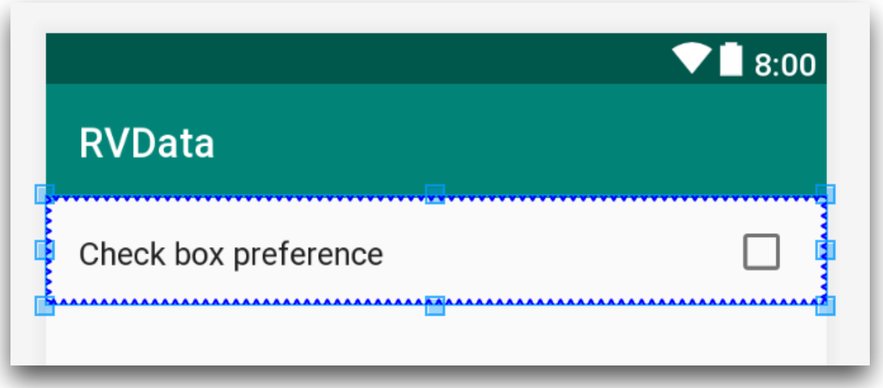

```xml
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

</PreferenceScreen>
```

Shared Preferences

- Preferences Palette
- Set the default value, key, title and (optional) summary




```
<CheckBoxPreference
    android:defaultValue="false"
    android:key="check_box_preference_1"
    android:title="Check box preference" />
```



Shared Preferences

- Subclass PreferenceFragmentCompat
- Set the default value, key, title and (optional) summary

```
public static class SettingsFragment extends PreferenceFragment {
    @Override
    public void onCreatePreferences(Bundle bundle, String s) {
        addPreferencesFromResource(R.xml.settings);
    }
}
```

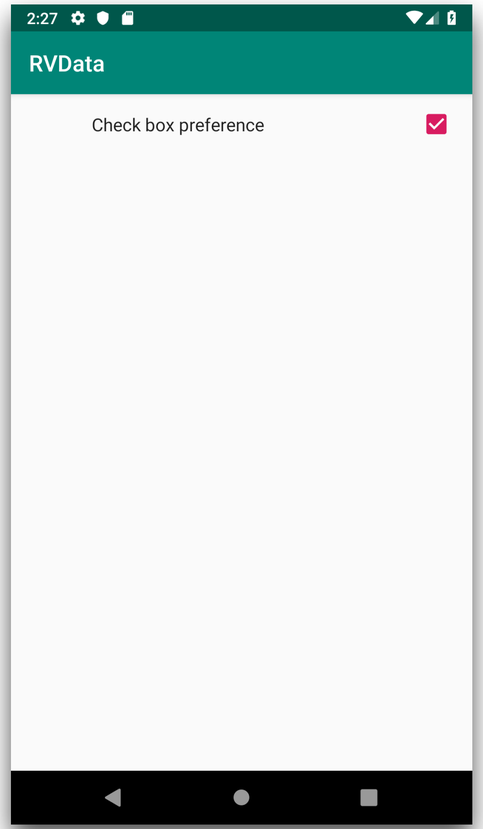
Shared Preferences

- In PreferenceFragmentCompat onCreate
- Add the Fragment

```
getSupportFragmentManager()
        .beginTransaction()
        .add(R.id.prefs_activity, new SettingsFragment())
        .commit();
```

Shared Preferences

- When necessary to show Settings, show via Intent.
- Changes will be saved and loaded each time.
- Fetch with Default preferences and keys defined in settings.xml




```
Intent intent = new Intent(this, PrefsActivity.class);
startActivity(intent);
```

```
SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(this);
Log.i("prefs", "prefs: " + preferences.getBoolean("check_box_preference_1", false));
```

Local Data Lab 2

1. Create a new Android Studio project.

2. Add the new dependency needed

3. Create new activity and basic layout and id

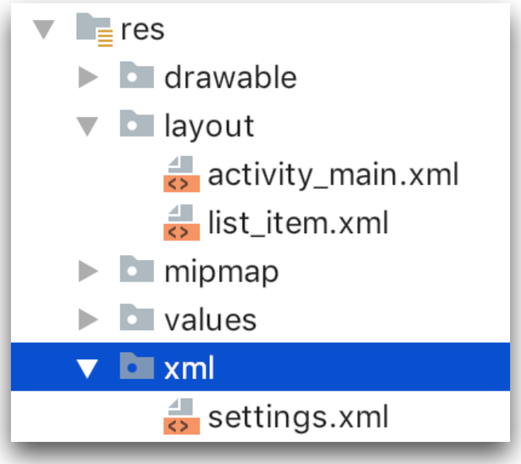
```
implementation 'androidx.preference:preference:1.0.0'
```

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    android:id="@+id/prefs_activity"
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".PrefsActivity">

</FrameLayout>
```

Lab 2

4. Create new res > xml directory

5. Create a new XML Resource File: settings.xml

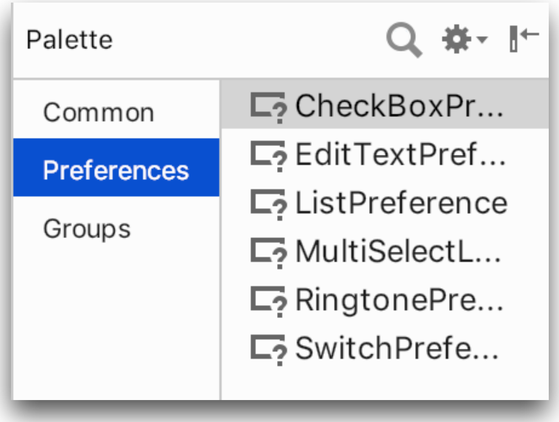



```
<?xml version="1.0" encoding="utf-8"?>
<PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android">

</PreferenceScreen>
```

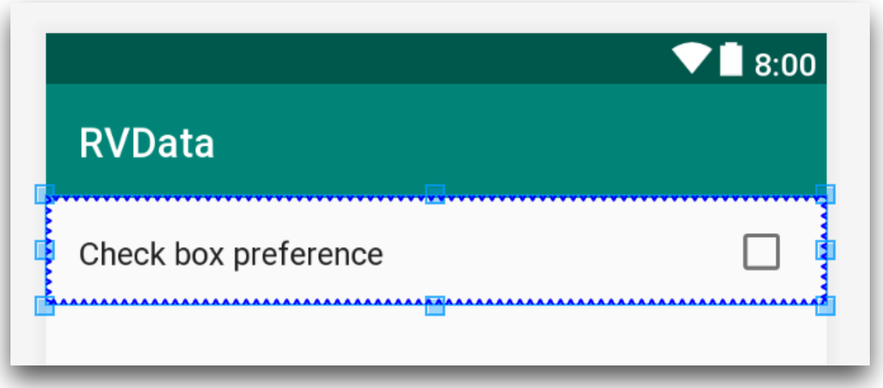
Lab 2

6. Edit the UI and add from the Preferences Palette

7. Set the default value, key, title and (optional) summary




```
<CheckBoxPreference
    android:defaultValue="false"
    android:key="check_box_preference_1"
    android:title="Check box preference" />
```



Lab 2

8. Subclass PreferenceFragmentCompat in the PrefsActivity class

9. Implement onCreatePreferences to load the preferences using settings.xml

```
public static class SettingsFragment extends PreferenceFragmentCompat {
    @Override
    public void onCreatePreferences(Bundle bundle, String s) {
        addPreferencesFromResource(R.xml.settings);
    }
}
```

Lab 2

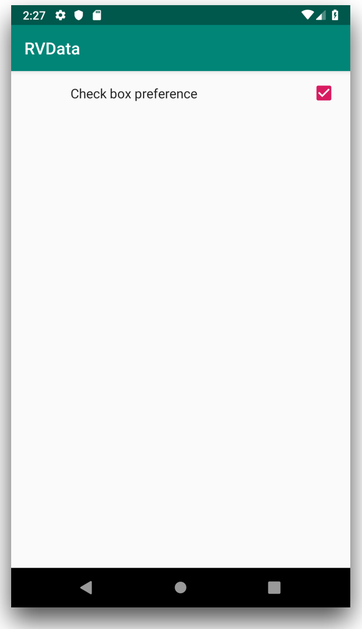
10. In the onCreate method for your Activity, add the Fragment

```
getSupportFragmentManager()
        .beginTransaction()
        .add(R.id.prefs_activity, new SettingsFragment())
        .commit();
```

Lab 2

11. Show via Intent (e.g., from MainActivity button tap)

12. Run and test.




```
Intent intent = new Intent(this, PrefsActivity.class);
startActivity(intent);
```

```
SharedPreferences preferences = PreferenceManager.getDefaultSharedPreferences(this);
Log.i("prefs", "prefs: " + preferences.getBoolean("check_box_preference_1", false));
```

Shared Preferences

- Classes can listen for SharedPreferences changes

```
private SharedPreferences.OnSharedPreferenceChangeListener prefsListener;
```

- OnSharedPreferencesChangeListener - create and register callback

```
@Override
public DataItemAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType)

    SharedPreferences settings =
            PreferenceManager.getDefaultSharedPreferences(mContext);
    prefsListener = new SharedPreferences.OnSharedPreferenceChangeListener() {
        @Override
        public void onSharedPreferenceChanged(SharedPreferences sharedPreferences,
                                              String key) {
            Log.i("preferences", "onSharedPreferenceChanged: " + key);
        }
    };
    settings.registerOnSharedPreferenceChangeListener(prefsListener);
```

Storage

Internal storage:

- It's always available.
- Files saved here are accessible by only your app.
- When the user uninstalls your app, the system removes all your app's files from internal storage.

Internal storage is best when you want to be sure that neither the user nor other apps can access your files.

External storage:

- It's not always available, because the user can mount the external storage as USB storage and in some cases remove it from the device.
- It's world-readable, so files saved here may be read outside of your control.
- When the user uninstalls your app, the system removes your app's files from here only if you save them in the directory from `getExternalFilesDir()`.

External storage is the best place for files that don't require access restrictions and for files that you want to share with other apps or allow the user to access with a computer.

Store Text File

- Internal Storage
- Get a directory as a File via either **getFilesDir**() (internal dir for your app) or **getCacheDir**() (cache for your app, can be deleted)
- Create a File via a File from one of the calls above and a name
- createTempFile to create in the cache directory

```
String filename = "data.txt";
File file = new File(this.getFilesDir(), filename);
if (file.exists()) {
    // operations
}
```

```
File cachefile = File.createTempFile("file.txt", null, this.getCacheDir());
```

Store Text File

- File object operations
  - canRead, canWrite, canExecute
  - delete(), exists(), compareTo(File)
  - See https://developer.android.com/reference/java/io/File

Internal Storage

- Write using an outputStream
- MODE_PRIVATE - private to your app, (see https://developer.android.com/training/secure-file-sharing/)

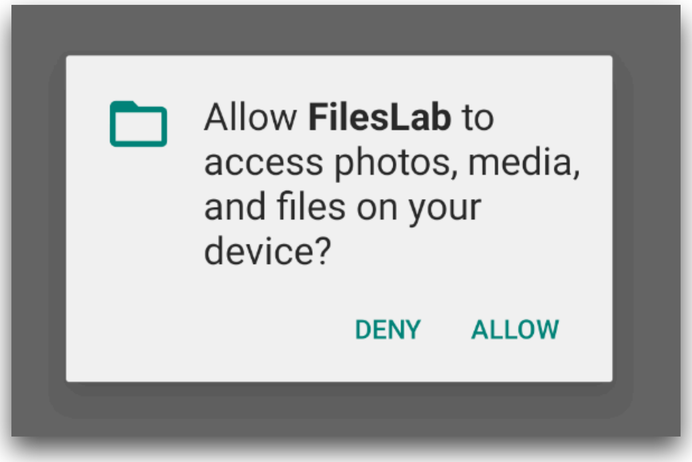
```
FileOutputStream outputStream;

try {
    outputStream = openFileOutput(filename, Context.MODE_PRIVATE);
    outputStream.write(fileContents.getBytes());
    outputStream.close();
} catch (Exception e) {
    e.printStackTrace();
}
```

```
private String readFile(Context context, String filename) {

    String ret = "";

    try {
        InputStream inputStream = context.openFileInput(filename);

        if ( inputStream != null ) {
            InputStreamReader inputStreamReader = new InputStreamReader(inputStream);
            BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
            String receiveString = "";
            StringBuilder stringBuilder = new StringBuilder();

            while ( (receiveString = bufferedReader.readLine()) != null ) {
                stringBuilder.append(receiveString);
            }

            inputStream.close();
            ret = stringBuilder.toString();
        }
    }
    catch (FileNotFoundException e) {
        Log.e("login activity", "File not found: " + e.toString());
    } catch (IOException e) {
        Log.e("login activity", "Can not read file: " + e.toString());
    }

    return ret;
}
```

External Storage

- Requires permissions in AndroidManifest and authorization from user

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



External Storage

- Check if external storage is writable/readable

```
public boolean isExternalStorageWritable() {
    String state = Environment.getExternalStorageState();
    return Environment.MEDIA_MOUNTED.equals(state);
}
```

```
public boolean isExternalStorageReadable() {
    String state = Environment.getExternalStorageState();
    return (Environment.MEDIA_MOUNTED.equals(state) ||
            Environment.MEDIA_MOUNTED_READ_ONLY.equals(state));
}
```

External Storage

- Check permissions and request authorization

```
// Initiate request for permissions.
private boolean checkPermissions() {

    if (!isExternalStorageReadable() || !isExternalStorageReadable()) {
        return false;
    }

    int permissionCheck = ContextCompat.checkSelfPermission(this,
            Manifest.permission.WRITE_EXTERNAL_STORAGE);
    if (permissionCheck != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(this,
                new String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE},
                REQUEST_PERMISSION_WRITE);
        return false;
    } else {
        return true;
    }
}
```

External Storage

- Handle response in onRequestPermissionsResult
- Store state

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       @NonNull String permissions[],
                                       @NonNull int[] grantResults) {
    switch (requestCode) {
        case REQUEST_PERMISSION_WRITE:
            if (grantResults.length > 0
                    && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
                permissionGranted = true;
            } else {
                permissionGranted = false;
            }
            break;
    }
}
```

External Storage

- Make a public directory (e.g., public Pictures)
- .mkdirs on File

```
public File getPublicAlbumStorageDir(String albumName) {
    // Get the directory for the user's public pictures directory.
    File file = new File(Environment.getExternalStoragePublicDirectory(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e("Storage", "Directory not created");
    }
    return file;
}
```

External Storage

- Get a private directory for the app

```
public File getPrivateAlbumStorageDir(Context context, String albumName) {
    // Get the directory for the app's private pictures directory.
    File file = new File(context.getExternalFilesDir(
            Environment.DIRECTORY_PICTURES), albumName);
    if (!file.mkdirs()) {
        Log.e("Storage", "Directory not created");
    }
    return file;
}
```

External Storage

- Get a private directory for the app

```
public File getExternalFile(String filename) {
    return new File(Environment.getExternalStorageDirectory(), filename);
}
```

- Write to it

```
public void writeToFile(File file, String fileContents) {
    try {
        FileOutputStream fos = new FileOutputStream(file);
        fos.write(fileContents.getBytes());
        fos.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

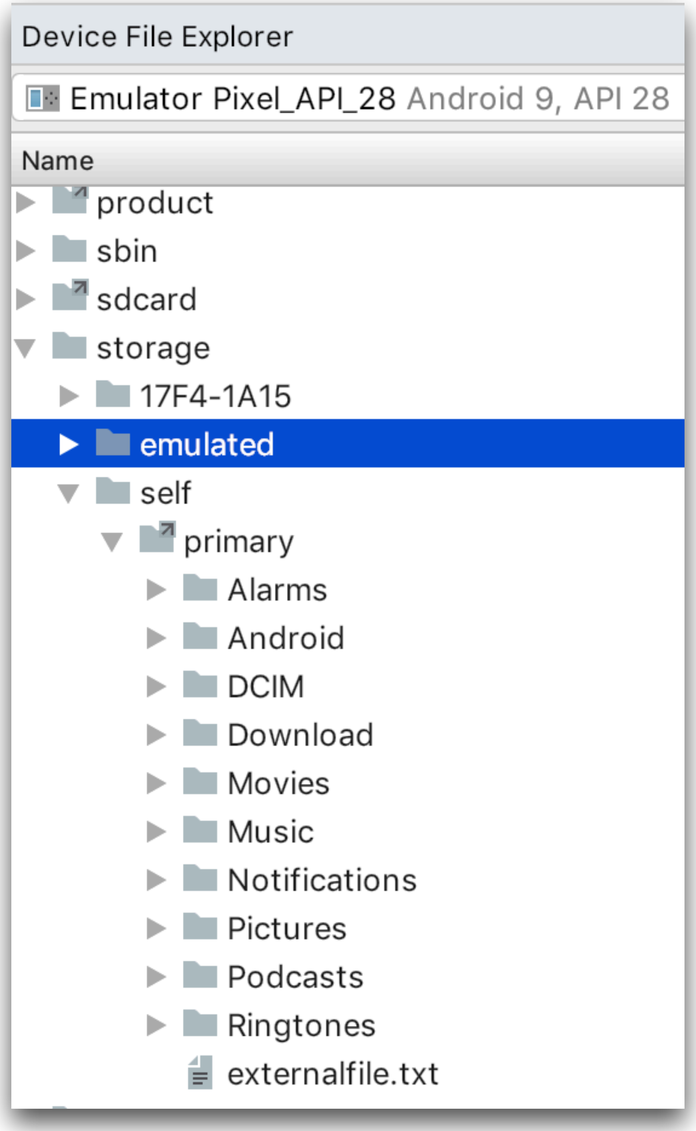
External Storage

- Read from a File object

```
public String readFromFile(File file) {
    String fileContent = "";
    String s = "";

    try {
        FileInputStream fIn = new FileInputStream(file);
        BufferedReader myReader = new BufferedReader(
                new InputStreamReader(fIn));

        while ((s = myReader.readLine()) != null) {
            fileContent += s + "\n";
        }
        myReader.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
    return fileContent;
}
```

External Storage

- See External files
- View > Tool Windows > Device File Explorer



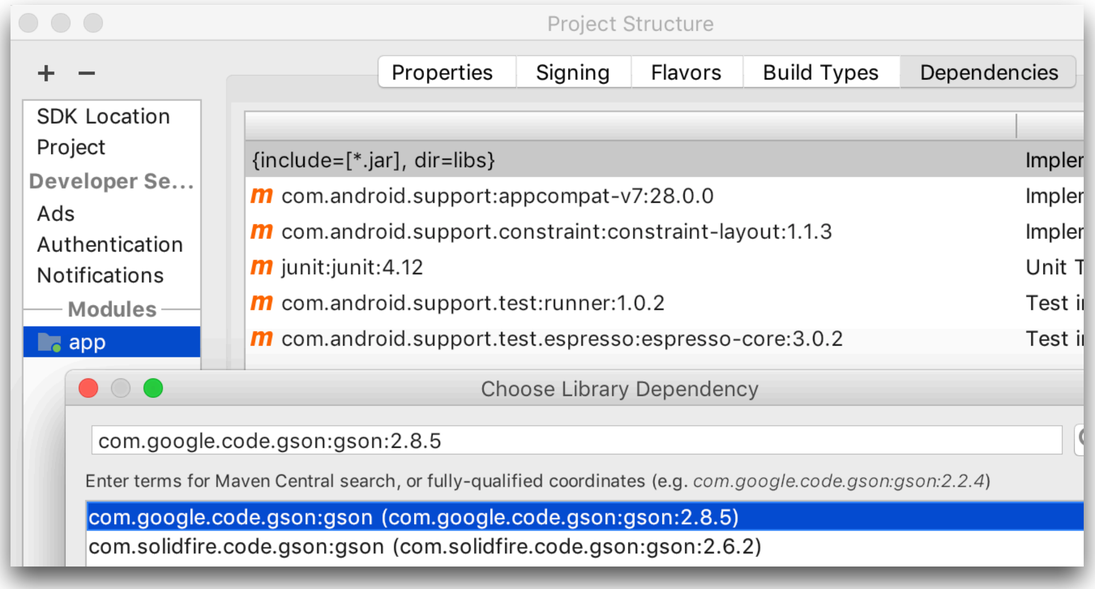
Local Data Lab Files Example

See LabFilesExample/FilesLab for example of the file-based operations.

JSON with Gson

- Add the Gson (from Google) dependency
- Needs to work with items and not collections so you may have to create a collection to how your items

```
private class DataItems {
    List<DataItem> dItems;

    public List<DataItem> getdItems() {
        return dItems;
    }

    public void setdItems(List<DataItem> dItems) {
        this.dItems = dItems;
    }
}
```



JSON with Gson

- Create Gson object and call toJson

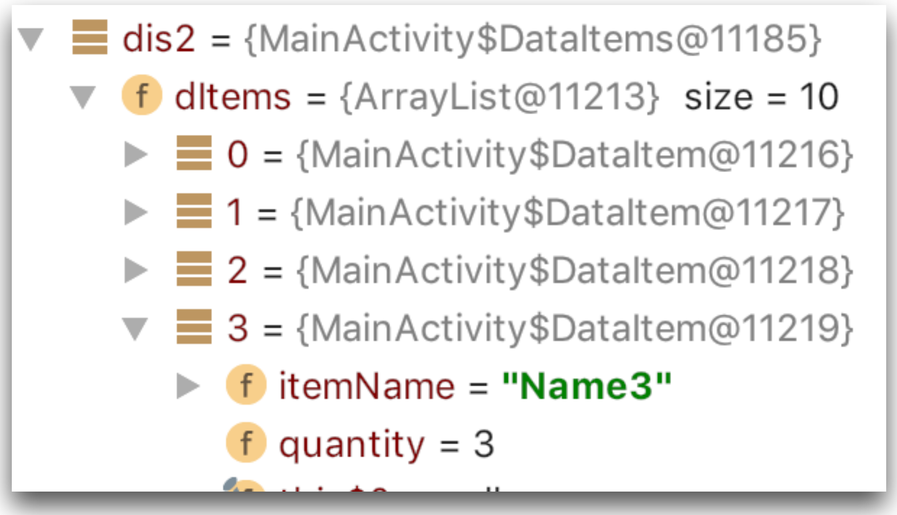
```
DataItems dis = new DataItems();
dis.dItems = items;

Gson gson = new Gson();
String jsonString = gson.toJson(dis);
```


```
{
   "dItems": [{
      "itemName": "Name0",
      "quantity": 0
   }, {
      "itemName": "Name1",
      "quantity": 1
   }, {
      "itemName": "Name2",
      "quantity": 2
   }, {
      "itemName": "Name3",
      "quantity": 3
   }, {
      "itemName": "Name4",
      "quantity": 4
   }, {
      "itemName": "Name5",
      "quantity": 5
   }, {
      "itemName": "Name6",
      "quantity": 6
   }, {
      "itemName": "Name7",
      "quantity": 7
   }, {
      "itemName": "Name8",
      "quantity": 8
   }, {
      "itemName": "Name9",
      "quantity": 9
   }]
}
```

JSON with Gson

- And back (from String to object)

```
DataItems dis2 = gson.fromJson(jsonString, DataItems.class);
```

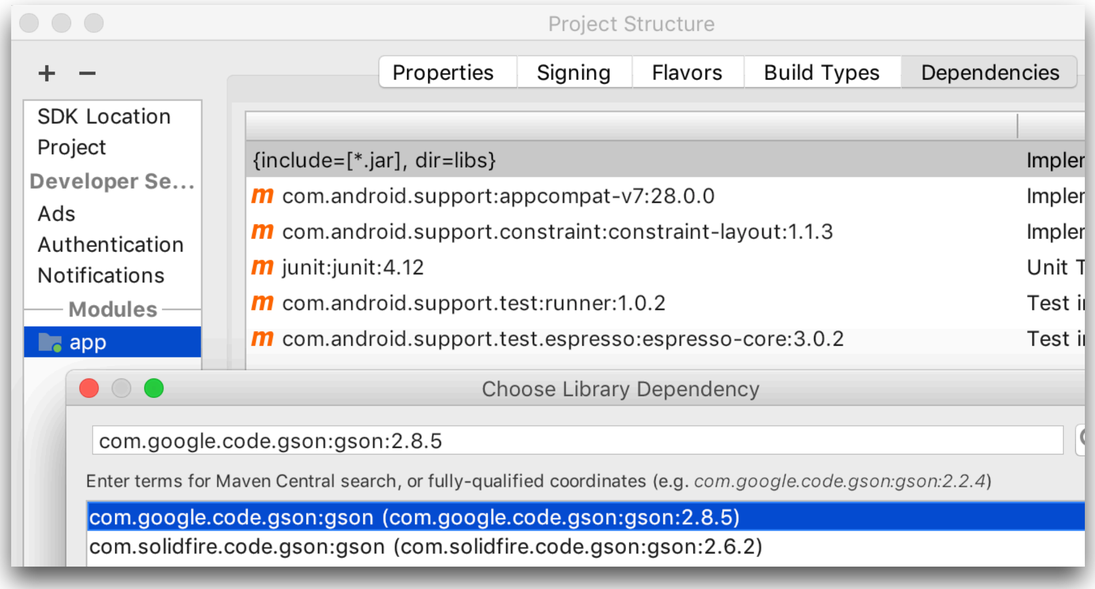


Local Data Lab 3

1. Open Lab3/JsonLabBegin project.

2. Add the Gson dependency (Google) and sync

3. Inspect the provided DataItem class



Lab 3

1. Create a new class called DataItems
2. Add a property that is List<DataItem> dItems
3. Use code generation to add a getter/setter

```
private class DataItems {
    List<DataItem> dItems;

    public List<DataItem> getdItems() {
        return dItems;
    }

    public void setdItems(List<DataItem> dItems) {
        this.dItems = dItems;
    }
}
```

Lab 3

1. Note the for loop that creates a List of DataItem instances in onCreate of MainActivity

2. Create an instance of DataItems after the loop

3. Set it's dItems property to the list created in the loop

4. Convert it to JSON using Gson

5. Run the app and test.

```
DataItems dis = new DataItems();
dis.dItems = items;

Gson gson = new Gson();
String jsonString = gson.toJson(dis);
```