CS 213 SOFTWARE METHODOLOGY

Lily Chang

CS Department @ Rutgers New Brunswick

FALL 2023

Android Views

Lecture Note #19

Android Views

- Every visible component on the screen is a View
- The View class represents the basic building block for user interface components, and the base class for classes that provide interactive UI components such as buttons, checkboxes, and text entry fields
- A View occupies a rectangular area on the screen and is responsible for drawing and event handling.
- A View has a location, expressed as a pair of left and top coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the density-independent pixel (dp)
- All the views in a window are arranged in a single tree. You can add views either from code or by specifying a tree of views in one or more XML layout files.

Android Views Relevant to Project 5

TextView

EditText / TextInputLayout

Button / ImageButton, onClick()

ImageView, onClick()

RadioButton

RadioGroup

CheckBox

Spinner

ListView / ObservableList

RecyclerView

TextView

- A user interface element that displays text to the user

```
<TextView
        android:id="@+id/text_view_id"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/hello"
/>
```

1. match_parent - as big as its parent (minus padding)
2. wrap_content - just big enough to enclose its content (plus padding).
3. exact values in dp (density-independent pixels)

xml attributes:
https://developer.android.com/reference/android/widget/TextView?hl=en

android:hint, andorid:gravity, android:textSize, android:onClick

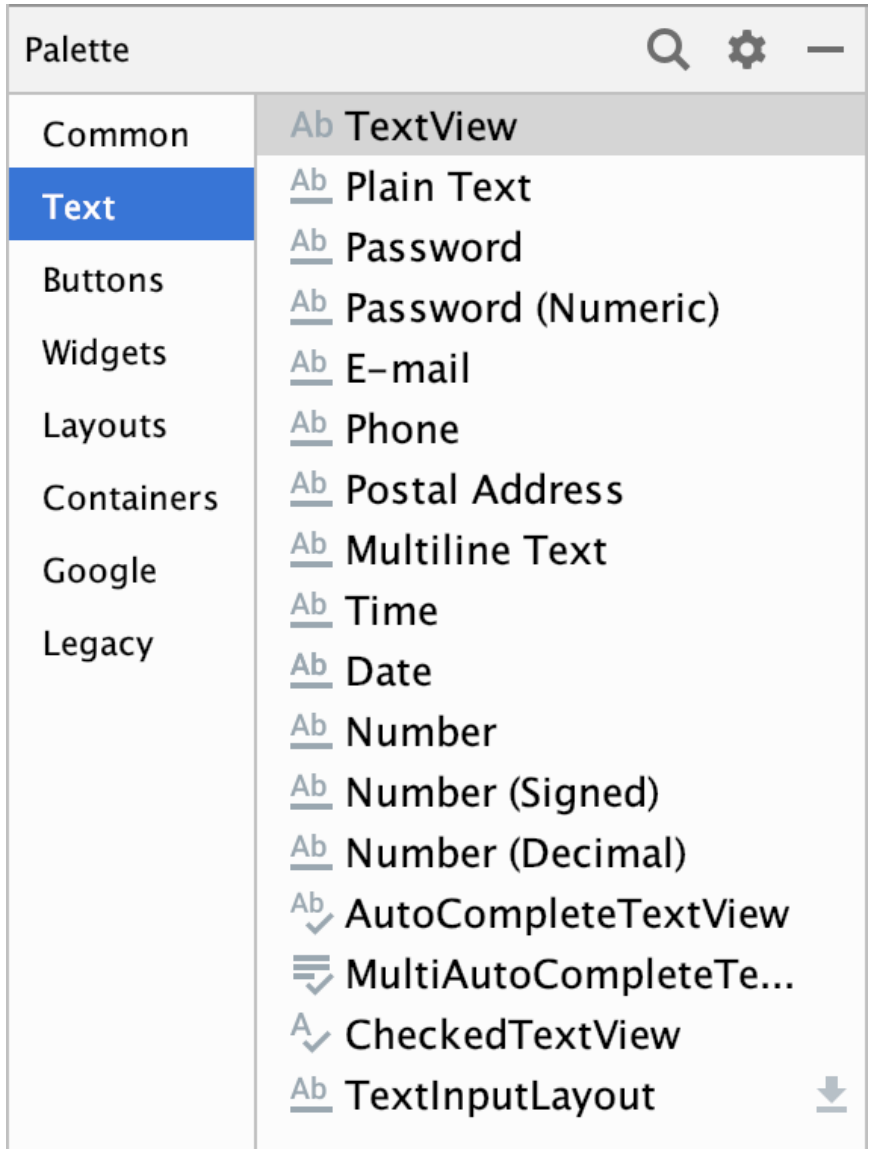
EditText

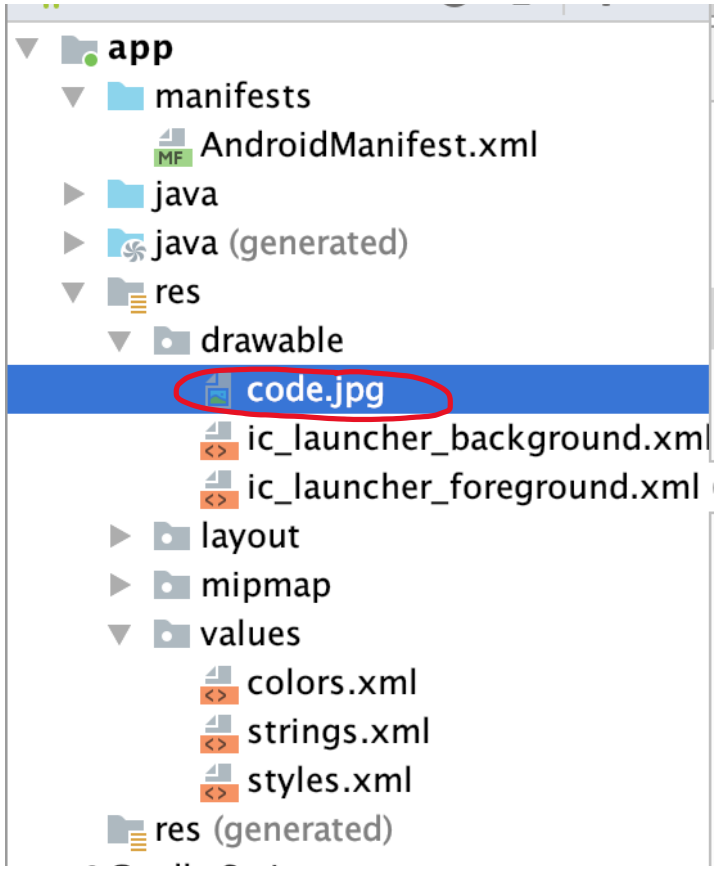
- A user interface element for entering and modifying text.
- Subclass of TextView
- When you define an edit text widget, you must specify the R.styleable.TextView_inputType attribute.
  - For example, for plain text input set inputType to "text"
  - Choosing the input type configures the keyboard type that is shown, acceptable characters, and appearance of the edit text.
  - For example, if you want to accept a secret number, like a unique pin or serial number, you can set inputType to "numericPassword”

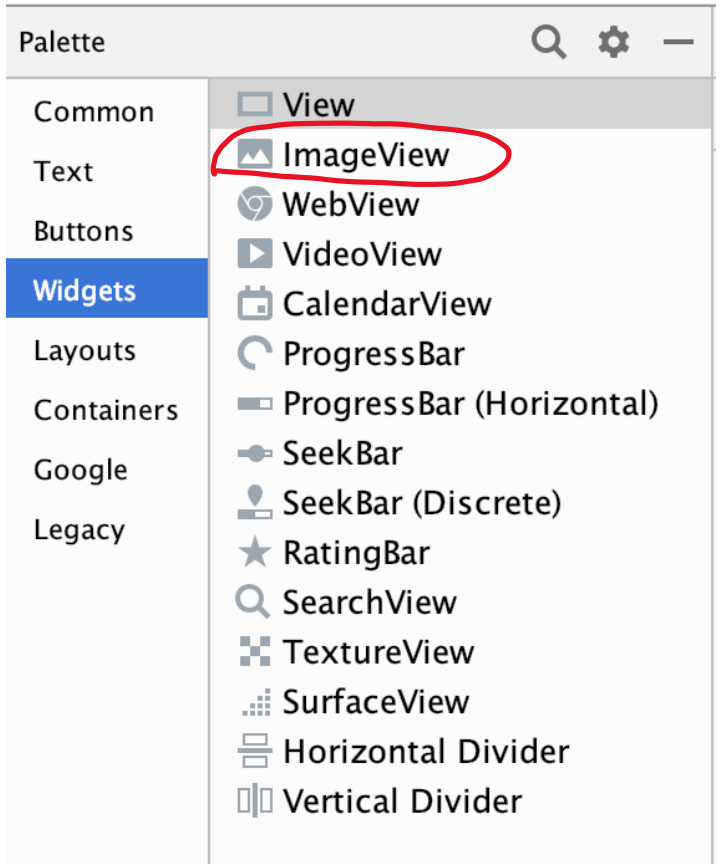
```
<EditText
  android:id="@+id/plain_text_input"
  android:layout_height="wrap_content"
  android:layout_width="match_parent"
  android:inputType="text”
/>
```

EditText

- Layout editor groups different EditText views into "Text" in the Palette
- You can choose different predefined Views based on the data needed to enter




app
manifests
AndroidManifest.xml
java
java (generated)
res
drawable
code.jpg
ic_launcher_background.xml
ic_launcher_foreground.xml
layout
mipmap
values
colors.xml
strings.xml
styles.xml
res (generated)


Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy
View
ImageView
WebView
VideoView
CalendarView
ProgressBar
ProgressBar (Horizontal)
SeekBar
SeekBar (Discrete)
RatingBar
SearchView
TextureView
SurfaceView
Horizontal Divider
Vertical Divider

ImageView

ImageView

- Displays image resources, for example Bitmap or Drawable resources
- ImageView is also commonly used to apply tints to an image and handle image scaling
- setImageResource() method sets a drawable as the content of this ImageView

  public void setImageResource(int resId)

```
<ImageView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:src="@drawable/my_image"
      android:contentDescription="@string/my_image_description"
/>
```

in res/drawable folder

R Class for accessing the resources

R.id

R.layout

R.drawable

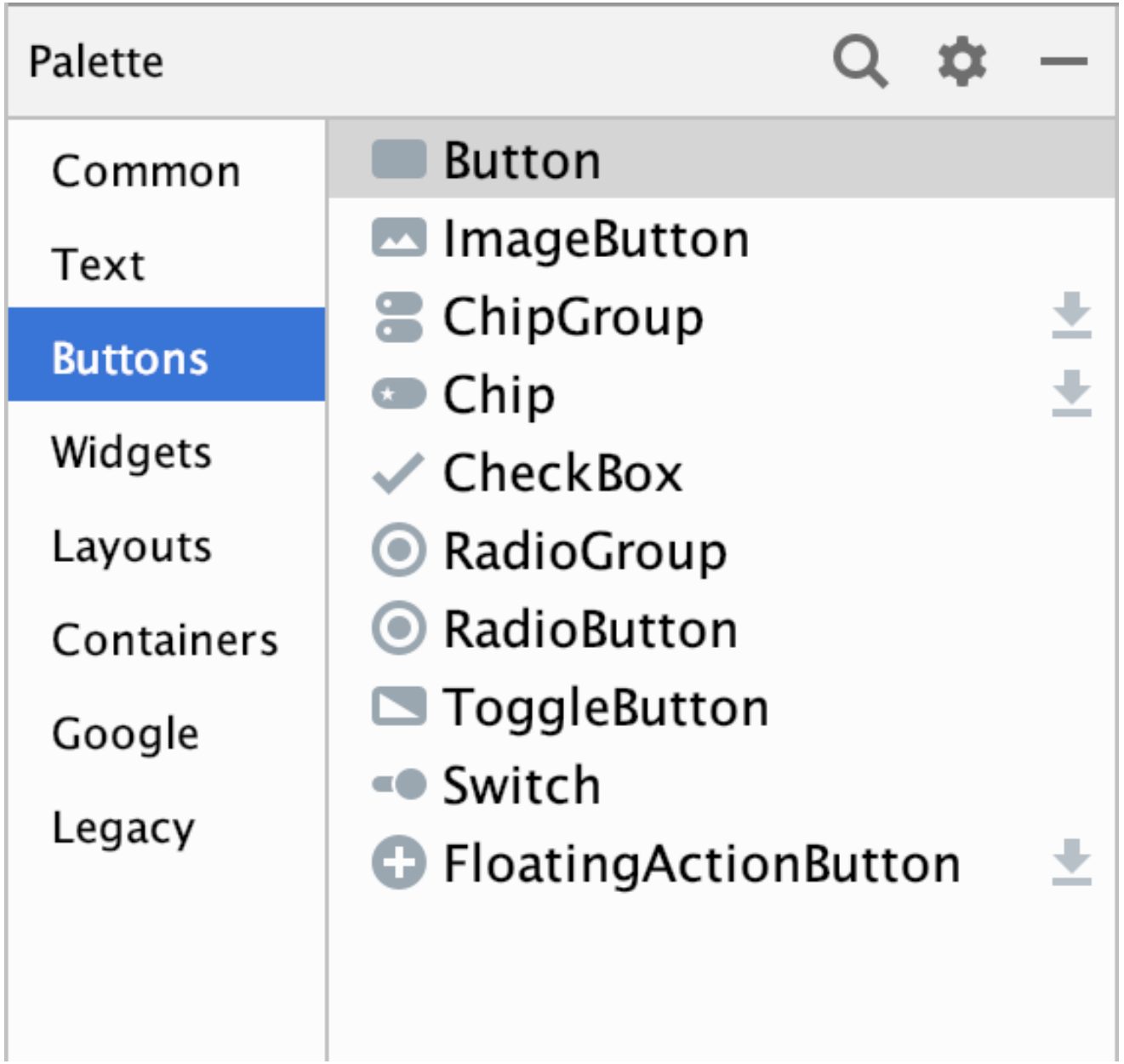
R.array

R.string

R.color

R.style

Buttons


Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy
Button
ImageButton
ChipGroup
Chip
CheckBox
RadioGroup
RadioButton
ToggleButton
Switch
FloatingActionButton

Buttons

RadioGroup

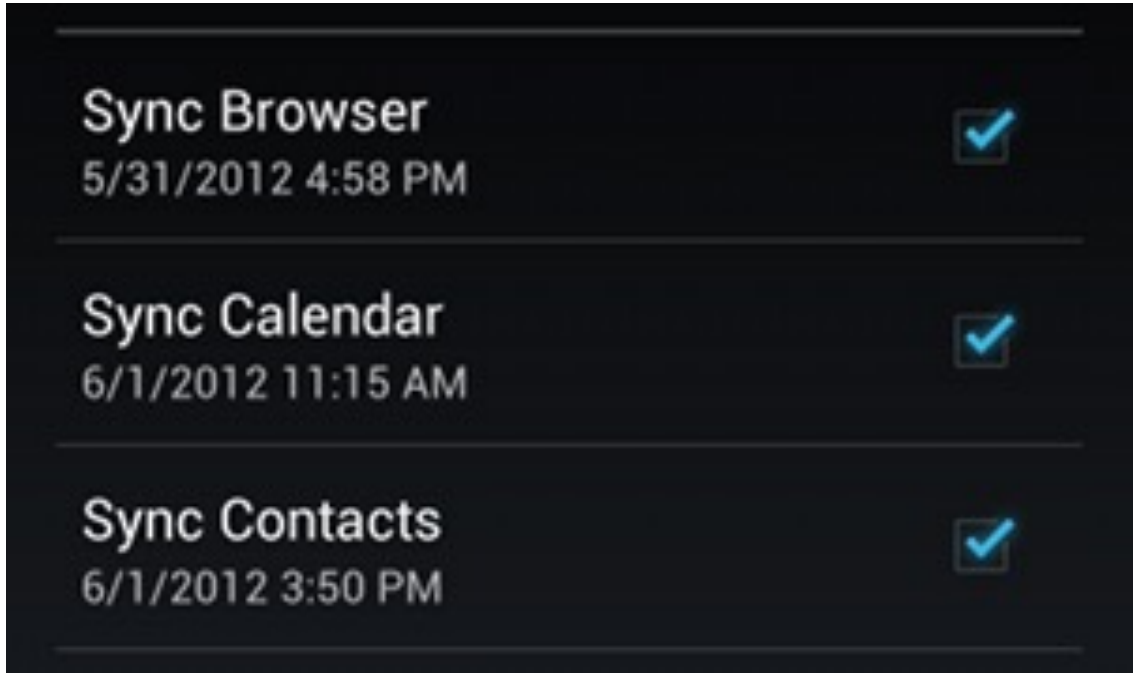
- This class is used to create a multiple-exclusion scope for a set of radio buttons. Checking one radio button that belongs to a radio group unchecks any previously checked radio button within the same group. Initially, all the radio buttons are unchecked.

RadioButton

- A radio button is a two-states button that can be either checked or unchecked.
- When the radio button is unchecked, the user can press or click it to check it. However, contrary to a CheckBox, a radio button cannot be unchecked by the user once checked.
- Radio buttons are normally used together in a RadioGroup. When several radio buttons live inside a radio group, checking one radio button unchecks all the others.
- Useful methods: **isChecked(), setChecked(), onClick()**

CheckBox

- A checkbox is a specific type of two-state button that can be either checked or unchecked.
- Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.
- When the user selects a checkbox, the CheckBox object receives an on-click event.
- Useful methods: isChecked(), setChecked(), onClick()



Adapter class

- An Adapter object acts as a bridge between an AdapterView and the underlying data for that view
- The Adapter provides access to the data items; it is also responsible for making a View for each item in the data set
- Subclasses of AdpaterView: ListView, Spinner, GridView, etc.
- AdapterView is a ViewGroup that displays items loaded into an adapter.
- The most common type of adapter comes from an array-based data source

ArrayAdapter

- Use ArrayAdapter to provide views for an AdapterView, such as ListView or Spinner; see useful methods here: https://developer.android.com/reference/android/widget/ArrayAdapter

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

ListView orders;
...
orders.setAdapter(orderlist); //show the data in the ListView
```

Spinner Class

- A view that displays one child at a time and lets the user pick among them (drop-down menu)
- The items in the Spinner come from the Adapter associated with this view
- Spinner is a subclass of the abstract class AdapterView
- If you define the data set with an array, use the ArrayAdapter
- Could set the reference to an array resource that will populate the Spinner with the attribute "android:entries" in the layout editor
- The simple_spinner_item layout and the simple_spinner_dropdown_item layout are the default predefined layouts provided by Android in the R.layout class

```
ArrayAdapter<CharSequence> adapter =
        ArrayAdapter.createFromResource(this, R.array.dataArray, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
mySpinner.setAdapter(adapter);
```

Spinner – Listener and Event-Handler

- When the user selects an item in the Spinner, the Spinner receives an on-item-selected event
- In order to handle the event, you must implement the Interface OnItemSelectedListner and override the 2 methods below; for example,

```
public class OrderActivity extends AppCompatActivity implements AdapterView.OnItemSelectedListener {
        private Spinner spinner;
        @Override
        protected void onCreate(Bundle savedInstanceState) {
      spinner = findViewById(R.id.mySpinner);
      spinner.setOnItemSelectedListener(this); //add the listener
    }
}
@Override
public void onItemSelected(AdapterView<?> parent, View view, int position, long id)  {
        //write code to handle the event
}
@Override
public void onNothingSelected(AdapterView<?> parent) { } //can leave it empty
```

ListView class

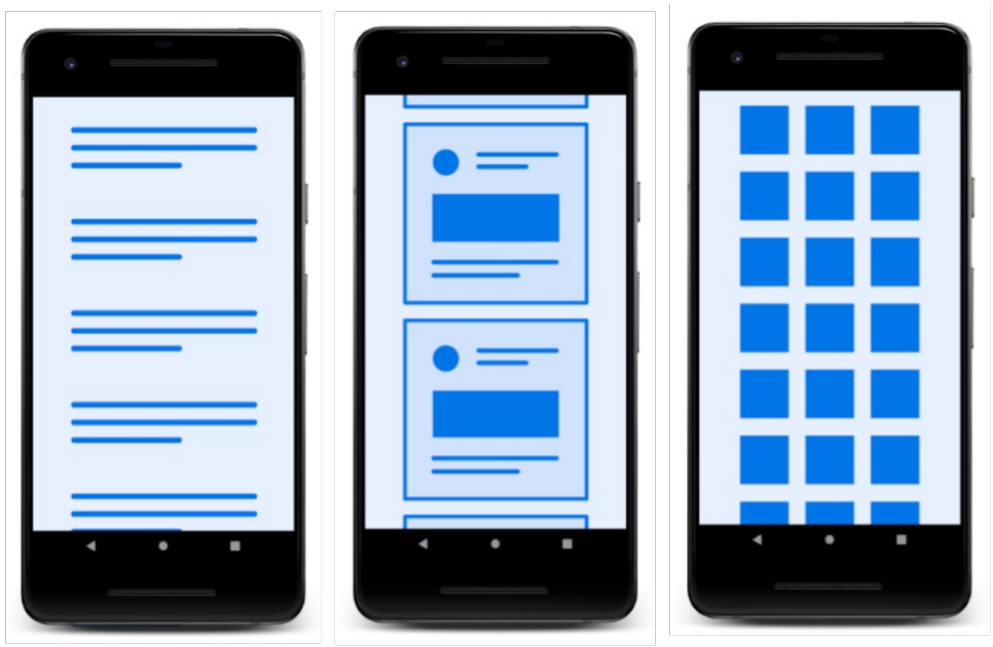
- Displays a vertically-scrollable collection of views, where each view is positioned immediately below the previous view in the list
- ListView class inherits the abstract class AdapterView (same as the Spinner)
- Set the data items with the AdapterView (same as the Spinner), or with the attribute “android:entries” in the layout editor
- When the user selects an item in the ListView, the ListView receives an on-item-click event
- In order to handle the event, you must implement the Interface OnItemClickListner and implement the onItemClick method and call the setOnItemClickListener(this) method

ListView / ObservableList

```
//must add the dependency androidx.databinding:databinding-runtime to build.gradle
public class MainActivity extends AppCompatActivity implements AdapterView.OnItemClickListener {
    private ListView listview;
    ObservableArrayList<String> list = new ObservableArrayList<>();
    String [] fruits = {"apple", "banana", "strawberry", "blueberry", "watermelon", "orange"};
    ArrayAdapter<String> items =
           new ArrayAdapter<String>(this, android.R.layout.simple_list_item_1, list);
    @Override
    protected void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        listView = findViewById(R.id.testList);
           Collections.addAll(list, fruits); //add objects to the ObservableList
        listView.setAdapter(items); //set the adapter of the ListView to the source
        listView.setOnItemClickListener(this); //add a listener to the ListView
    }
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id)
    { //write code to handle the event }
```

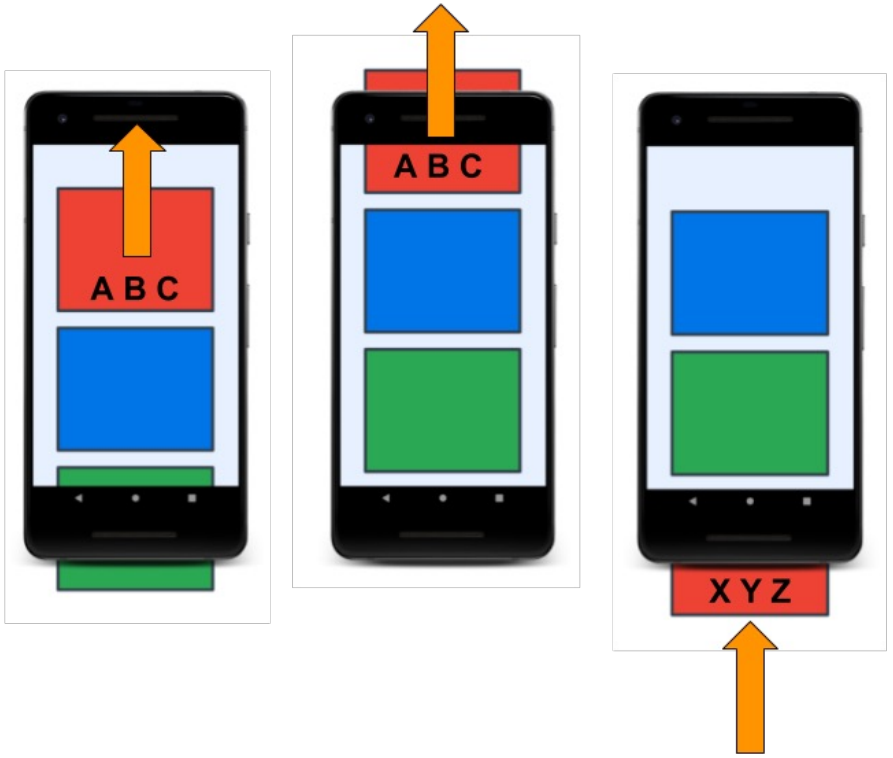
RecyclerView

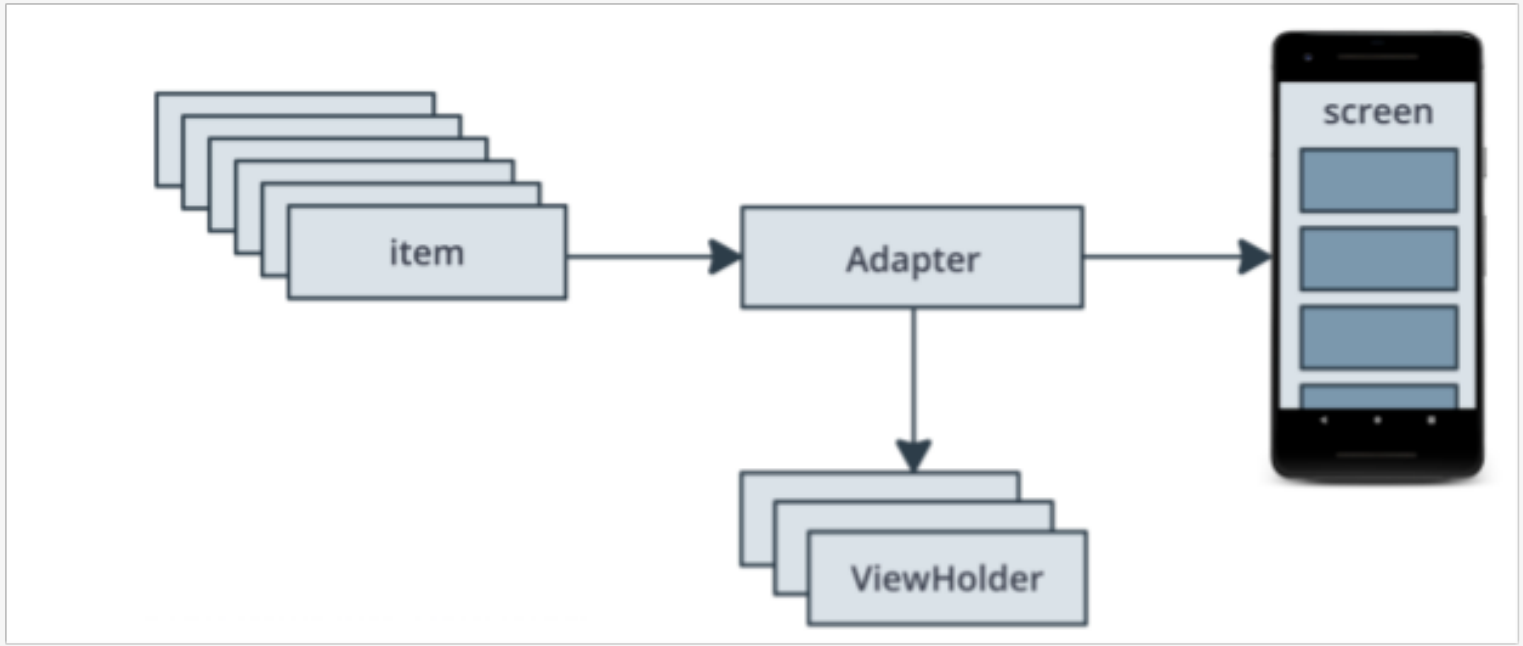
- Displaying a list or grid of data is one of the most common UI tasks in Android.
- Lists vary from simple to very complex.
- A list of text views might show simple data, such as a shopping list.
- A complex list, such as an annotated list of vacation destinations, might show the user many details inside a scrolling grid with headers.



RecyclerView

- By default, RecyclerView only does work to process or draw items that are currently visible on the screen. For example, if your list has a thousand elements but only 10 elements are visible, RecyclerView does only enough work to draw 10 items on the screen.
- When the user scrolls, RecyclerView figures out what new items should be on the screen and does just enough work to display those items.
- When an item scrolls off the screen, the item's views are recycled. That means the item is filled with new content as it scrolls onto the screen. This RecyclerView behavior saves a lot of processing time and helps lists scroll smoothly.
- When an item changes, instead of redrawing the entire list, RecyclerView can update that one item. This is a huge efficiency gain when displaying long lists of complex items!





Implementing A Recyclerview

Implementing a RecyclerView

- To display your data in a RecyclerView, you need the following parts:
  - Data to display.
  - A RecyclerView instance defined in your layout file, to act as the container for the views.
  - A layout for one item of data (a row on the screen.) If all the list items look the same, you can use the same layout for all of them, but that is not mandatory. The item layout has to be created separately from the fragment's layout, so that one item view at a time can be created and filled with data.
  - A layout manager. The layout manager handles the organization (the layout) of UI components in a view.
  - A view holder. The view holder extends the ViewHolder class. It contains the view information for displaying one item from the item's layout. View holders also add information that RecyclerView uses to efficiently move views around the screen.
  - An adapter. The adapter connects your data to the RecyclerView. It adapts the data so that it can be displayed in a ViewHolder. A RecyclerView uses the adapter to figure out how to display the data on the screen.
  - See the sample code posted on Canvas

Steps for implementing a RecyclerView

- First, add a RecyclerView into your layout the way you would add any other UI element
- Next, design how each item in the list is going to look and behave
  - ✓ Decide the source of data, or create a class for each item
  - ✓ Create a layout for each item
  - ✓ Each individual element in the list is defined by a view holder object defined in the Adapter
- Create an Adapter class by extending RecyclerView.Adapter
  - ✓ The RecyclerView requests those views, and binds the views to their data, by calling methods in the adapter
  - ✓ Define the view holder by extending RecyclerView.ViewHolder
  - ✓ When the view holder is created, it doesn't have any data associated with it
  - ✓ After the view holder is created, the RecyclerView binds it to its data
- The items in your RecyclerView are arranged by a LayoutManager class
  - ✓ The layout manager arranges the individual elements in your list

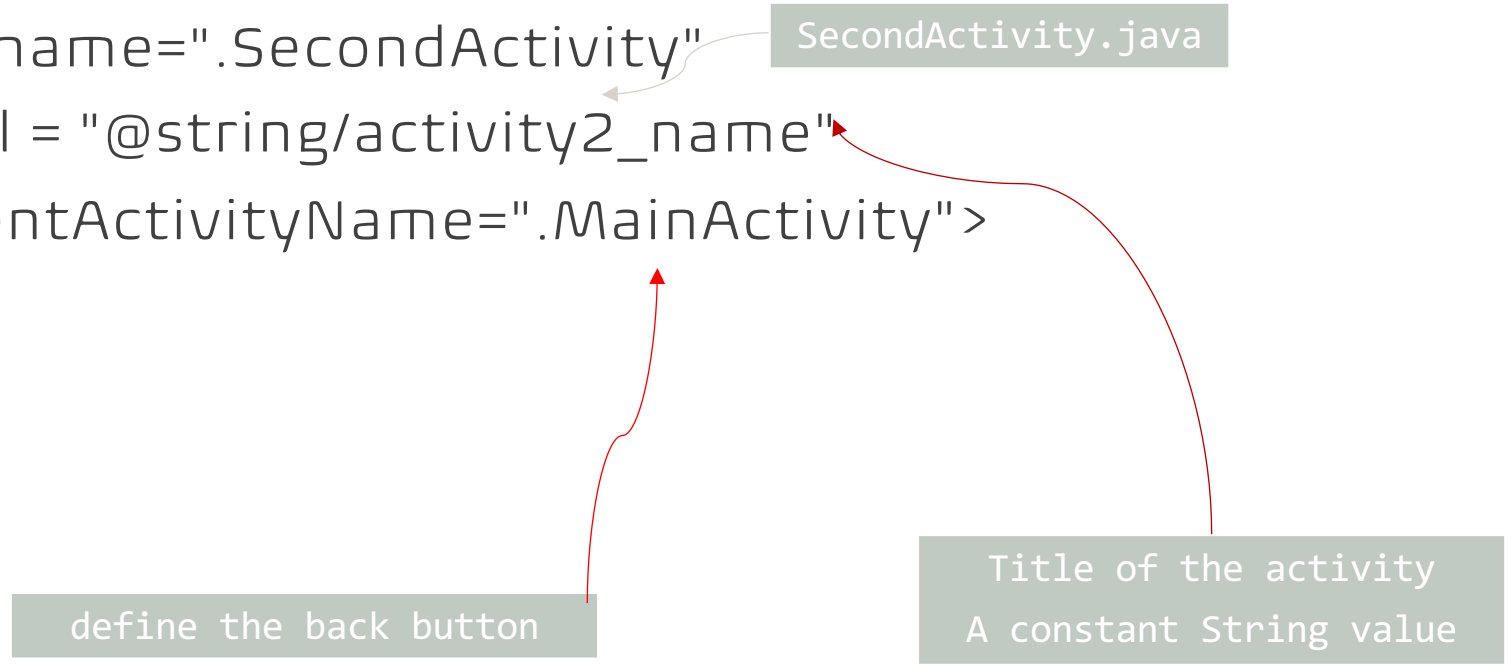
Activity Class

AndroidManifest.xml	Each activity has a .java file and a .xml file; for example,	The .xml file is where you create the UI components

• Register the activities in your app
• Define the Intent Filter
  • Explict intent
  • Implict intent
	• MainActivity.java → activity_main.xml
	• SecondActivity.java → activity_second.xml
		• Use the Android Layout Editor
		• Write the xml code directly

Adding an Activity in AndroidManifest.xml

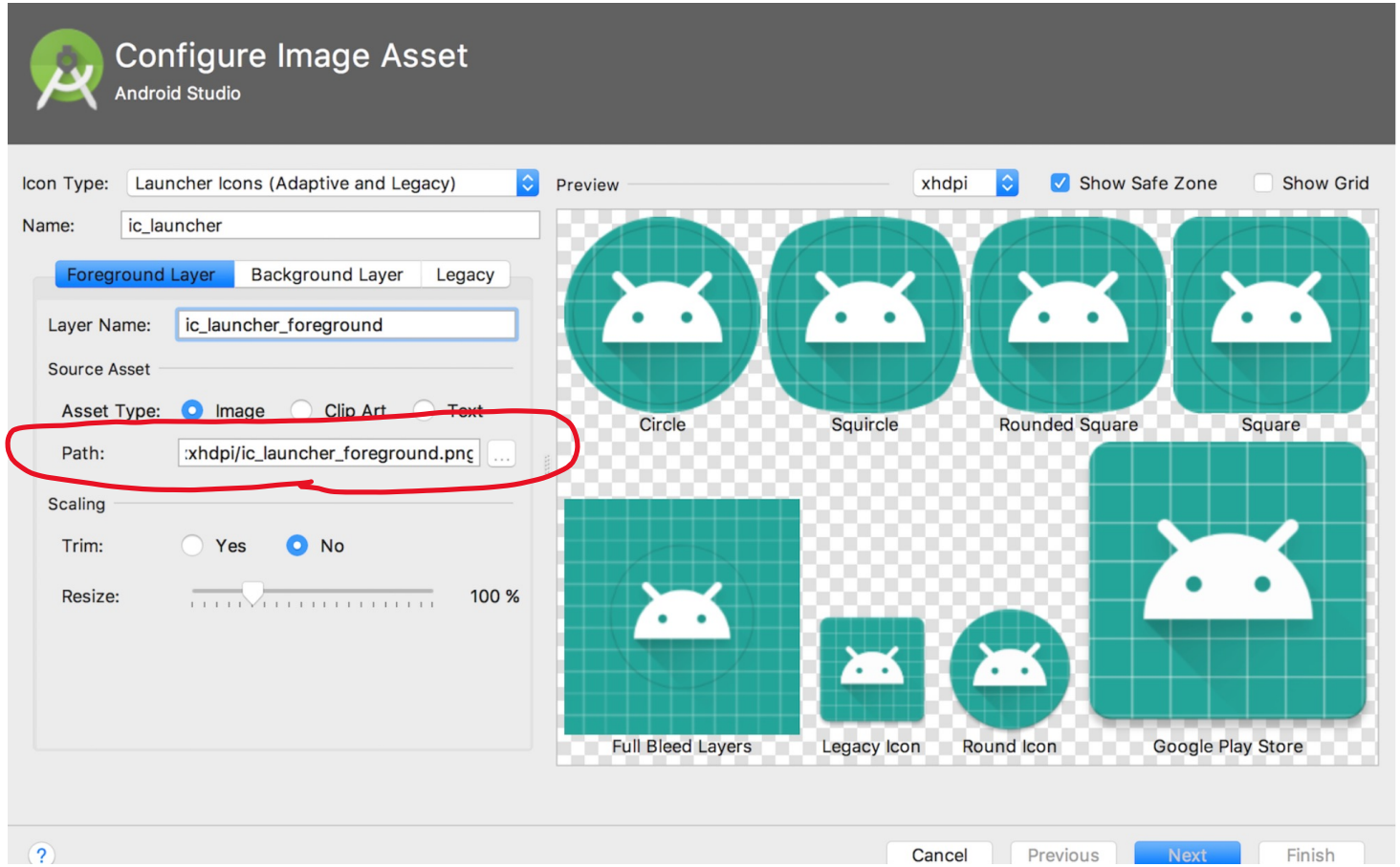
```
<activity android:name=".SecondActivity"
        android:label = "@string/activity2_name"
        android:parentActivityName=".MainActivity">
</activity>
```



Launcher Icon

A launcher icon is a graphic that represents your app to users. It can:

- Appear in the list of apps installed on a device and on the Home screen.
- Represent shortcuts into your app (for example, a contact shortcut icon that opens detail information for a contact).
- Be used by launcher apps.
- Help users find your app on Google Play.
- Right-click the **res** folder and select **New** > **Image Asset**.
- Path: browse to the folder where you have the image file you wanted to use as the launcher icon



Share Data Between Android Activities

Remember that Android Activities are running in their own sandboxes. There are different ways to share data/objects between Activities.

1. Use Intent extra (one way only) to encapsulate the data in the Intent object.
2. Store data persistently in external files on local drive or to a remote storage, which can be accessed by different Activities.
3. Create a subclass of Application class
4. Use a Singleton class (Singleton Design Pattern)
5. define public static variables

Intent Object Revisited

- The Intent object specifies either the exact activity you want to start or describes the type of action you want to perform (and the system selects the appropriate activity for you, which can even be from a different application).
- An Intent object can also carry small amounts of data to be used by the activity that is started.

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

Intent Class/Object

- The primary pieces of information in an intent are:
- action – The general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.
- data – the data to operate on, such as a person record in the contacts database, expressed as a Uri.
- For example,

```
ACTION_VIEW content://contacts/people/1
    //Display information about the person whose identifier is ”1".
```

Intent Extra

Sharing data between different activities

This is a **Bundle** of any additional information.

This can be used to provide extended information to the component.

For example, if we have an action to send an e-mail message, we could also include extra pieces of data here to supply a subject, body, etc.

```
public final class Bundle
    extends BaseBundle implements Cloneable, Parcelable
```

java.lang.Object
↳ android.os.BaseBundle
↳ android.os.Bundle

A mapping from String keys to various `Parcelable` values.

Warning: Note that `Bundle` is a lazy container and as such it does NOT implement `Object.equals(java.lang.Object)` or `Object.hashCode()`.

Sending Extra Data

- You can send data when starting another Activity

public Intent putExtra (String name, Bundle value)

✓ name – String: the name of the extra data; used as a key to retrieve the value.

✓ value – Bundle: the Bundle data value; this value may be null.

The key name to be used to retrieve the Bundle value. By convention, use the Package prefix so it is unique

value could be String, primitive types or class types that implement Serializable or Parcelable Interface

```
//in MainActivity
public static final String EXTRA_MESSAGE = "com.example.android.twoactivities.extra.MESSAGE"; //the key name
String message = "……..";
intent.putExtra(EXTRA_MESSAGE, message);
startActivity(intent); //this will start the activity and sent the extra message
```

```
//in SecondActivity
Intent intent = getIntent();
//retrieve the value with the key name
String message = intent.getStringExtra(FirstActivity.EXTRA_MESSAGE);
```

Intent Extra – Integer type

```
//in MainAtivity
int size = 10;
Intent intent = new Intent(this, SecondActivity.class);
//key name is MAXSIZE, value is size
intent.putExtra(“MAXSIZE”, size);

//in SecondActivity
Intent intent = getIntent();
int size = intent.getIntExtra(“MAXSIZE”, 0);
```

default value is 0 if the extra data is not available for some reason.

Intent Extra – Class type

- For the object you wanted to send, the class type must implement either Serializable Interface or Parcelable Interface (preferred)
- Serialization method; the class type must **implement the Serializable Interface** (no additional code needed)

```
//MainActivity
Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra(”ORDER", order); //where order is an instance of Order class that is serialized
startActivity(intent);
```

```
//SecondActivity
Intent intent = getIntent();
//retrieve and deserialize the extra data
Order order = (Order) intent.getSerializableExtra(”ORDER");
```

Intent Extra – Parcel

- Part of the Android SDK
- Preferred method for sending objects when starting an activity
- Class type must implement Parcelable Interface to send the "parcel" object; there are a few methods you need to implement, so additional coding is needed
- See the Adroid API reference for implementation https://developer.android.com/reference/android/os/Parcelable
- Use the follow method to get the parcel object in the SecondActivity

```
public T getParcelableExtra (String keyName)
```

Example – Parcelable

```
//MainActivity

intent.putExtra(”ORDER", order);

//SecondActivity
Intent intent = getIntent();
Order order =
intent.getParcelableExtra(”ORDER");
```

```
public class MyParcelable implements Parcelable {
    private int mData;

    public int describeContents() {
        return 0;
    }

    public void writeToParcel(Parcel out, int flags) {
        out.writeInt(mData);
    }

    public static final Parcelable.Creator<MyParcelable> CREATOR
            = new Parcelable.Creator<MyParcelable>() {
        public MyParcelable createFromParcel(Parcel in) {
            return new MyParcelable(in);
        }

        public MyParcelable[] newArray(int size) {
            return new MyParcelable[size];
        }
    };

    private MyParcelable(Parcel in) {
        mData = in.readInt();
    }
}
```

Application class

It is the Base class for maintaining global application state.

You can provide your own implementation by creating a subclass and specifying the fully-qualified name of this subclass as the "android:name" attribute in your AndroidManifest.xml's <application> tag.

The Application class, or your subclass of the Application class, is instantiated before any other class when the process for your application/package is created.

If you want to maintain data that are globally accessible in your app, create a subclass extends the Application class, and provide the getter and setter methods in this subclass

There is normally no need to subclass Application. In most situations, static singletons can provide the same functionality in a more modular way.

Example – define a subclass of Application

- Define a class to extend the Application class, see the coding example below
- Add the attribute android:name=".GlobalData" under the <application> tag in the AndroidManifest.xml

```
import android.app.Application
public final class GlobalData extends Application {
    private String data2share; //the global data
    public void setData(String data) { //setter
        data2share = data;
    }
    public String getData() { //getter
        return data2share;
    }
}
//the code below in some Activity can access the global
data
GlobalData globalData =
        (GlobalData) getApplicationContext(); //get the reference
globalData.setData("…"); //update the data with the setter
String data  = globalData.getData(); //get the data with the getter
```

Example – define a Singleton class

```
public final class GlobalData {
    private static GlobalData globalData; //single instance
    private String data2share; //global data to share
    private GlobalData() { } //so JVM will not do anything
    public static synchronized GlobalData getInstance() {
        if (globalData == null)
            globalData = new GlobalData(); //lazy approach
        return globalData;
    public void setData(String data) { //setter
        data2share = data;
    }
    public String getData() { //getter
        return data2share;
    }
}
//the code below can be used in any Activity to access the data
GlobalData globalData = GlobalData.getInstance();
globalData.setData("…"); //update the data with the setter
String data = globalData.getData(); //get the data with the getter
```

What is Gradle?

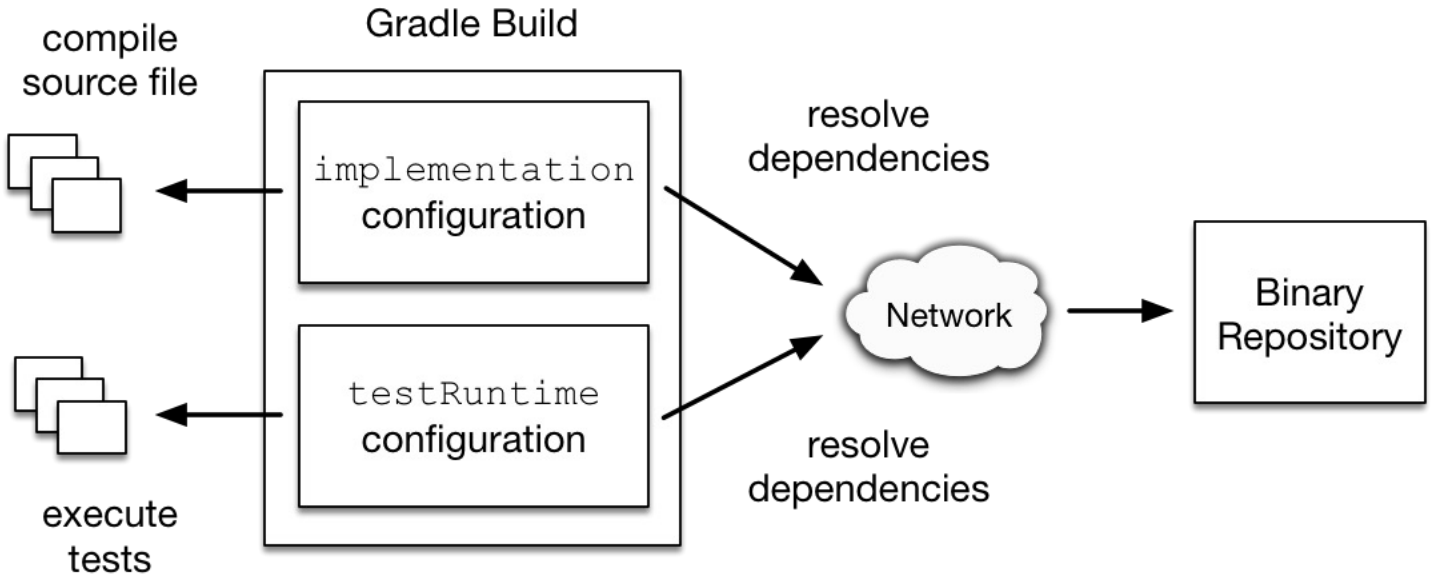
- Gradle is an open-source build automation tool that is designed to be flexible enough to build almost any type of software
- Avoids unnecessary work by only running the tasks that need to run because their inputs or outputs have changed
- Runs on the JVM and you must have a Java Development Kit (JDK) installed to use it.
- You can readily extend Gradle to provide your own task types or even build model
- Build scans provide extensive information about a build run that you can use to identify build issues.

App/Build.Gradle

```
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.3"

    defaultConfig {
        applicationId "com.example.demoandroid"
        minSdkVersion 24
        targetSdkVersion 30
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {...}
}

dependencies {

    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
}
```

What are dependency configurations?

- Every dependency declared for a Gradle project applies to a specific scope.
- For example, some dependencies should be used for compiling source code whereas others only need to be available at runtime.
- Gradle represents the scope of a dependency with the help of a Configuration. Every configuration can be identified by a unique name
- Module dependencies, File dependencies, Project dependencies



Material Design UI Components

Make sure you include the material design dependencies in build.gradle

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
}
```