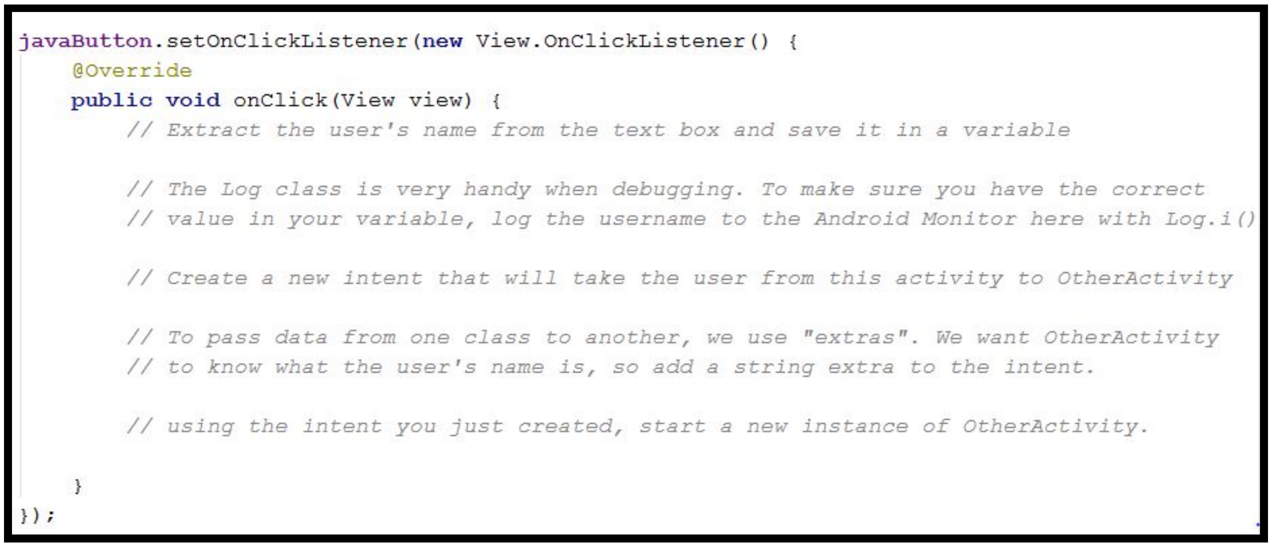
Android Beginners’ Session Three

Handling Events

In this lesson, we will explore the ways you can handle common events in Android such as what to do when a button is clicked, transitioning from one screen to another, and some debugging tips. The end of this guide will explain more advanced concepts such as the process of connecting to the internet with asynchronous tasks.

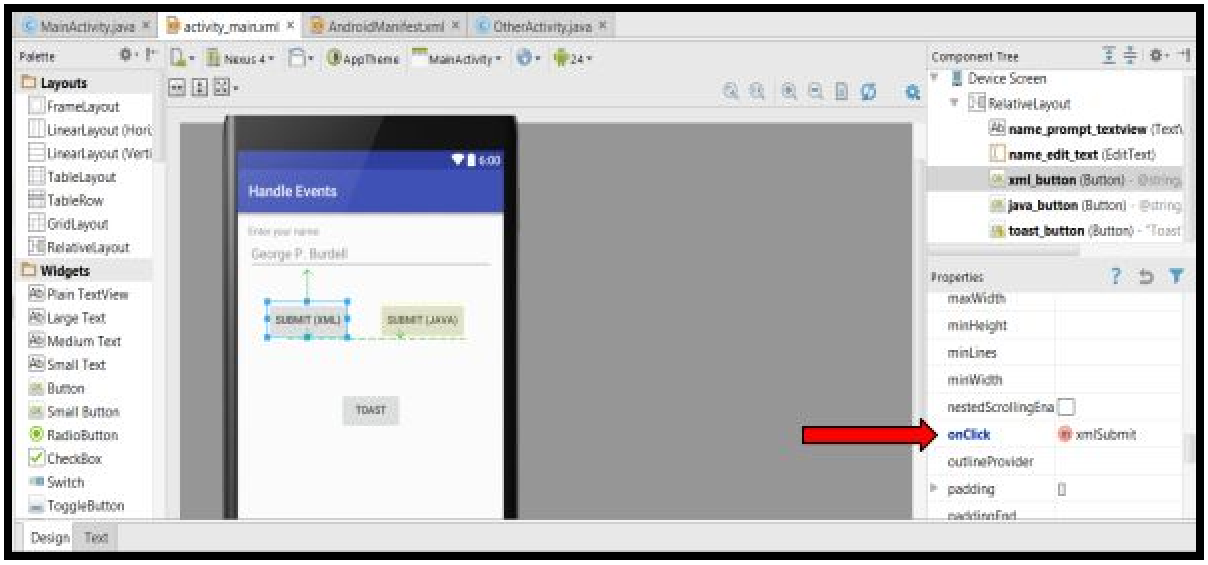
OnClickListeners

To bring the user interface (UI) to life, we need a way of letting the various UI elements know how to respond when the user clicks on them. This is where the concept of a listener comes in. A listener is an object that partners with a UI element and just listens for click events. When the listener “hears” a click, it will tell its UI partner what to do. As a developer, you will need to override the onClick method of your listener. Here is an example of pairing a listener with a button in java:



```java
javaButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Extract the user's name from the text box and save it in a variable

        // The Log class is very handy when debugging. To make sure you have the correct
        // value in your variable, log the username to the Android Monitor here with Log.i()

        // Create a new intent that will take the user from this activity to OtherActivity

        // To pass data from one class to another, we use "extras". We want OtherActivity
        // to know what the user's name is, so add a string extra to the intent.

        // using the intent you just created, start a new instance of OtherActivity.

    }
});
```

Android also allows you to use the XML design palette to take control of click events. If you already have a method defined that corresponds to the expected behavior of a UI element upon click, you can link that method directly to the UI element’s “onClick” property (example shown in blue below)



onClick xmlSubmit

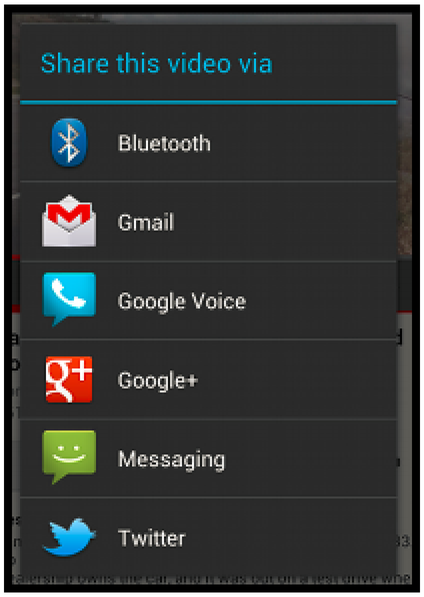
Using the XML design palette is a less common approach to handling click events, but it is something to be aware of.

Starting Activities

To move from one activity to another, you need to define an intent that tells the Android OS what you *intend* on doing. These come in two main forms: explicit and implicit. Explicit intents are useful when you know exactly what to do such as moving from one activity to another within your app. In the example below, we want to move from the current activity (HomePageActivity) to a new activity (ItemListActivity). We tell the Android OS that we want to go specifically from *this* instance of HomePageActivity (give the intent context as to where you are) to a new instance of the ItemListActivity *class* (give the intent context as to where you want to go).

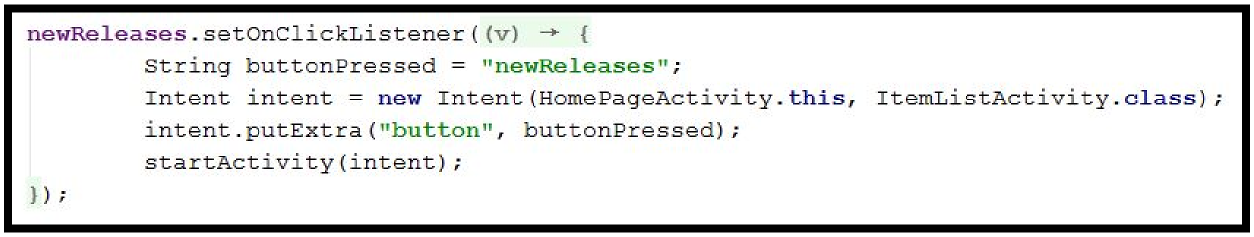
```java
newReleases.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent intent = new Intent(HomePageActivity.this, ItemListActivity.class);
        startActivity(intent);
    }
});
```

Implicit intents are useful when you know you want to do something, but are not sure of how the user would like it to be done. Let's say your app lets you send a message to someone. You want to ask the user which app they want to use to send the message. The code is similar to that of the explicit intent, but you just need to add the intent action and force the app chooser to appear (see the intent link for more info). Once the implicit intent is implemented, this is what it will look like to a user:

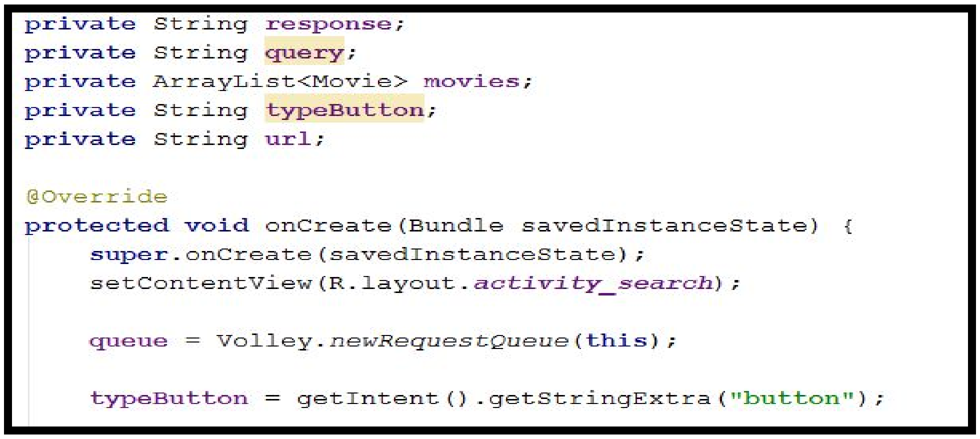


Passing Data Between Activities

As the app moves from one activity to another, there will often be times where you want the app to “remember” something during the transition. Luckily, the same intents that start new activities can also carry *extra* data along with them. Intent extras operate similarly to maps/dictionaries in that they use key-value pairs to store and retrieve data. Think of moving to a new city. You need to pack your belongings (putExtra), move (startActivity), and then unpack (getExtra) when you reach your destination. In the example below, we use the value in buttonPressed (HomePageActivity) to define a variable, typeButton, located in ItemListActivity as we transition from HomePageActivity to ItemListActivity.



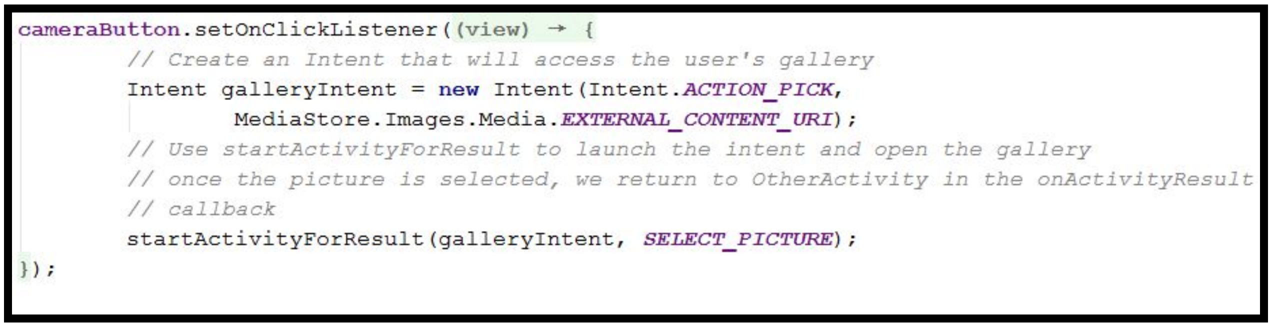
```
newReleases.setOnClickListener((v) → {
        String buttonPressed = "newReleases";
        Intent intent = new Intent(HomePageActivity.this, ItemListActivity.class);
        intent.putExtra("button", buttonPressed);
        startActivity(intent);
});
```



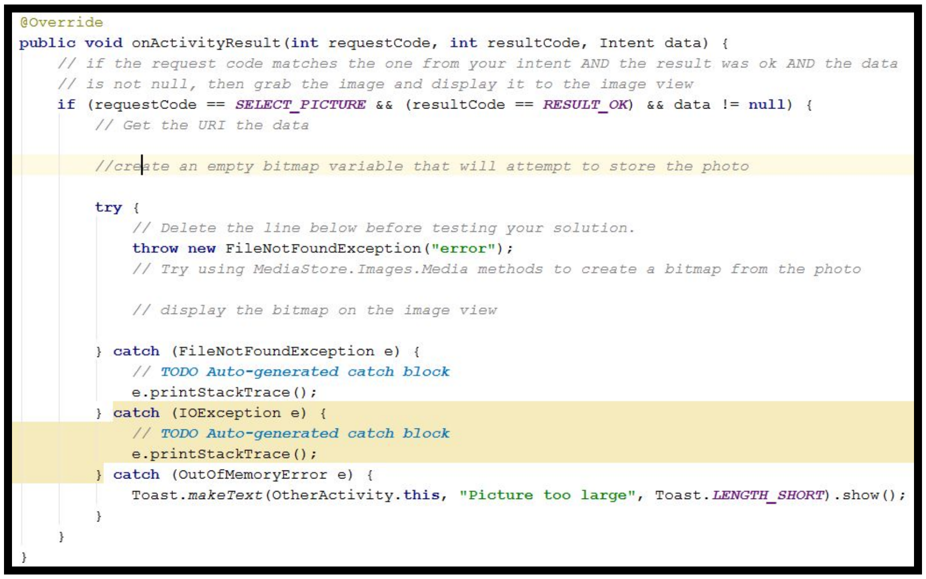
```
private String response;
private String query;
private ArrayList<Movie> movies;
private String typeButton;
private String url;

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

    queue = Volley.newRequestQueue(this);

    typeButton = getIntent().getStringExtra("button");
```

Getting Results From Intents

When a temporary transition from an activity is needed, an intent can be setup in such a way that allows you to move to another application, complete some task, and return to your application once the task is completed. A common example is allowing the user to take a picture and attach it somewhere in your app. The user leaves your app, accesses the camera app to take a picture, and returns to your app with the picture. To do this, we have to start a new activity for a specific result. When the activity completes its task, it will return to the original activity via the onActivityResult callback. In this method, you can handle the data that was returned (i.e. setting an image from the camera to an ImageView). An example is shown below.



```
cameraButton.setOnClickListener((view) → {
        // Create an Intent that will access the user's gallery
        Intent galleryIntent = new Intent(Intent.ACTION_PICK,
                MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
        // Use startActivityForResult to launch the intent and open the gallery
        // once the picture is selected, we return to OtherActivity in the onActivityResult
        // callback
        startActivityForResult(galleryIntent, SELECT_PICTURE);
});
```

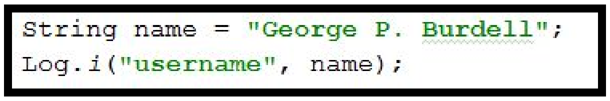


```java
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
    // if the request code matches the one from your intent AND the result was ok AND the data
    // is not null, then grab the image and display it to the image view
    if (requestCode == SELECT_PICTURE && (resultCode == RESULT_OK) && data != null) {
        // Get the URI the data

        //create an empty bitmap variable that will attempt to store the photo

        try {
            // Delete the line below before testing your solution.
            throw new FileNotFoundException("error");
            // Try using MediaStore.Images.Media methods to create a bitmap from the photo

            // display the bitmap on the image view

        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (OutOfMemoryError e) {
            Toast.makeText(OtherActivity.this, "Picture too large", Toast.LENGTH_SHORT).show();
        }
    }
}
```

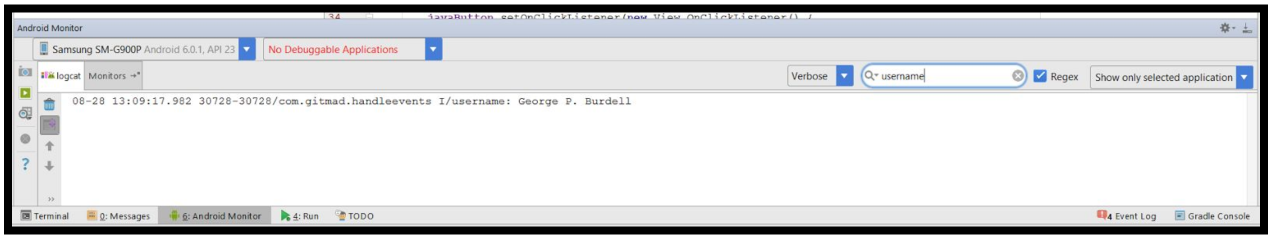
For more information about getting results from activities, read about them here.

Debugging

When handling events in Android, sometimes, things do not execute as planned. Fortunately, Android has some handy tools that help us debug and see what's going on under the hood. The first tool we'll discuss is the Android Log. If you click the Android Monitor at the bottom of Andorid Studio, you will be able to see logs of all the events happening in Android. This is helpful because you can use this to check the values of variables to make sure assignments are occuring as expected. In the example below, we want to log the result of assigning a string to the variable "name". The tag "username" is used to filter through all of the events happening. Searching the log by tag will display only results that have been tagged with a particular string.

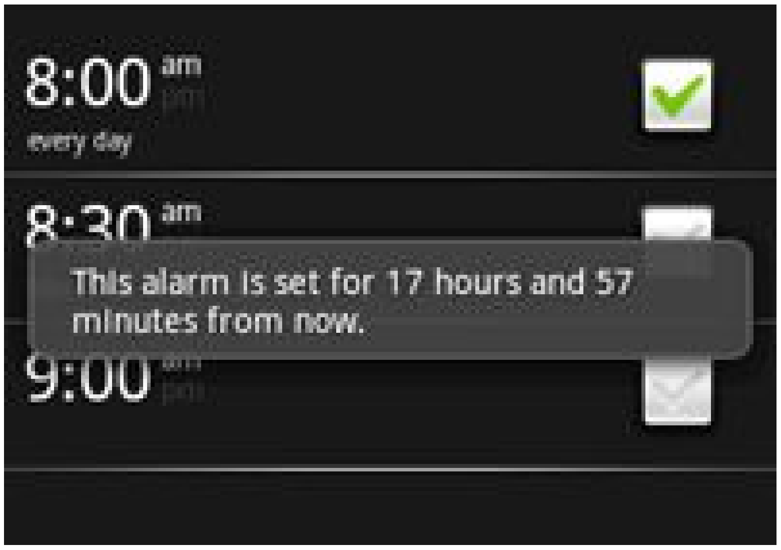


```java
String name = "George P. Burdell";
Log.i("username", name);
```



```
08-28 13:09:17.982 30728-30728/com.gitmad.handleevents I/username: George P. Burdell
```

A second helpful technique for debugging is Toast creation. A toast is simply a message displayed on the UI to alert or confirm something for the user. We can use this in a similar way that we use the log statements by passing the "name" variable as the toast's text and see if the expected string appears. An example of a toast is shown below.



This alarm is set for 17 hours and 57 minutes from now.

Advanced: Background Tasks

Have you ever wanted to be in two places at one? Android apps often have to do this to complete a task while occupying the user's attention. If there is ever an operation that utilizes a lot of CPU resources (i.e. a quick download), it cannot be done on the main UI thread. So far, everything that we have been doing with layouts and activities have occurred on the UI thread because that is where the user spends most of their time. If a download were to occur on the UI thread, we would see slowing of visual effects and "freezing" until the download completes. To get around this obstacle, Android allows us to create an *asynchronous task* that operates independently of the UI thread via doInBackground. Now, we can perform actions such as a quick download while serving the user smooth UI interactions. Be advised that AsyncTasks are like a light version of a thread and are only recommended for short term events. If you need long term background operations, create a new thread and handler. Once the background operation completes, the AsyncTask's onPostExecute method kicks in on the UI thread so you may use your recent download with the UI.

Note: when extending from the AsycTask class, you need to indicate three type parameters → AsyncTask<1,2,3>

1. This type corresponds to the type of var args you will give to doInBackground
2. This type corresponds to the type of var args you will give to onProgressUpdate
3. This type corresponds to the type of variable passed in to onPostExecute which is also the same as the return type for doInBackground.