Asynchronous Programming in Android

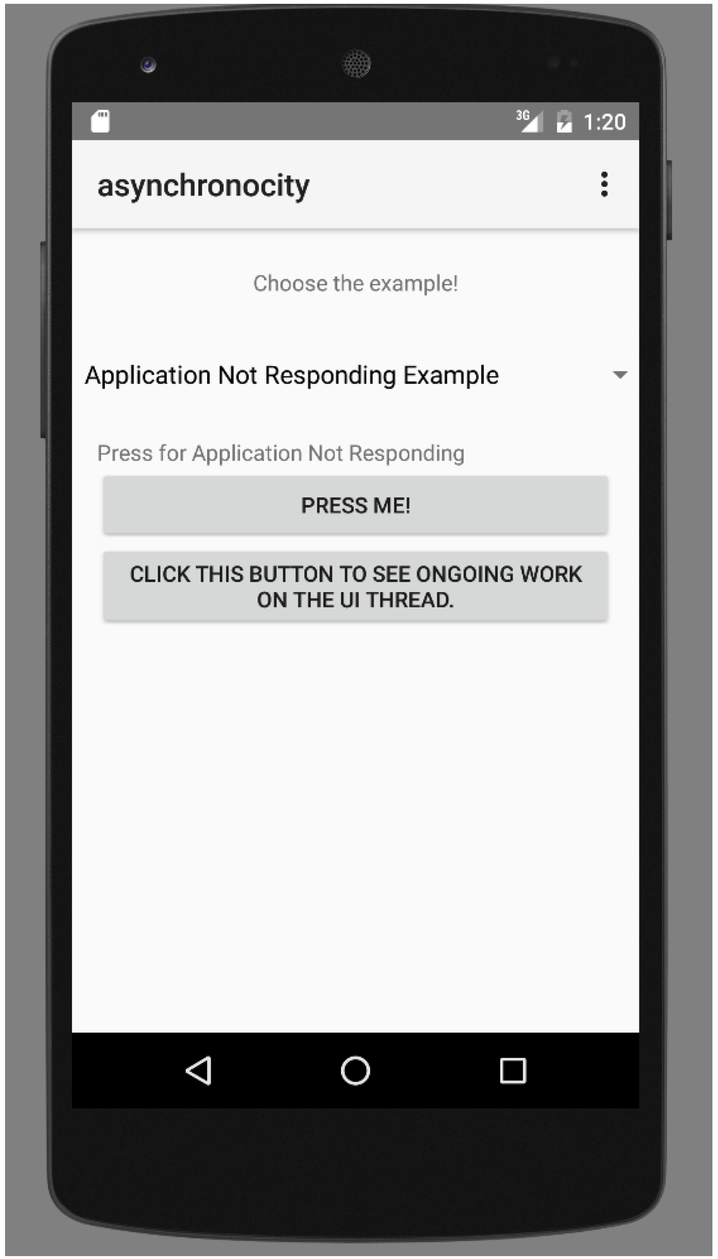
About Me

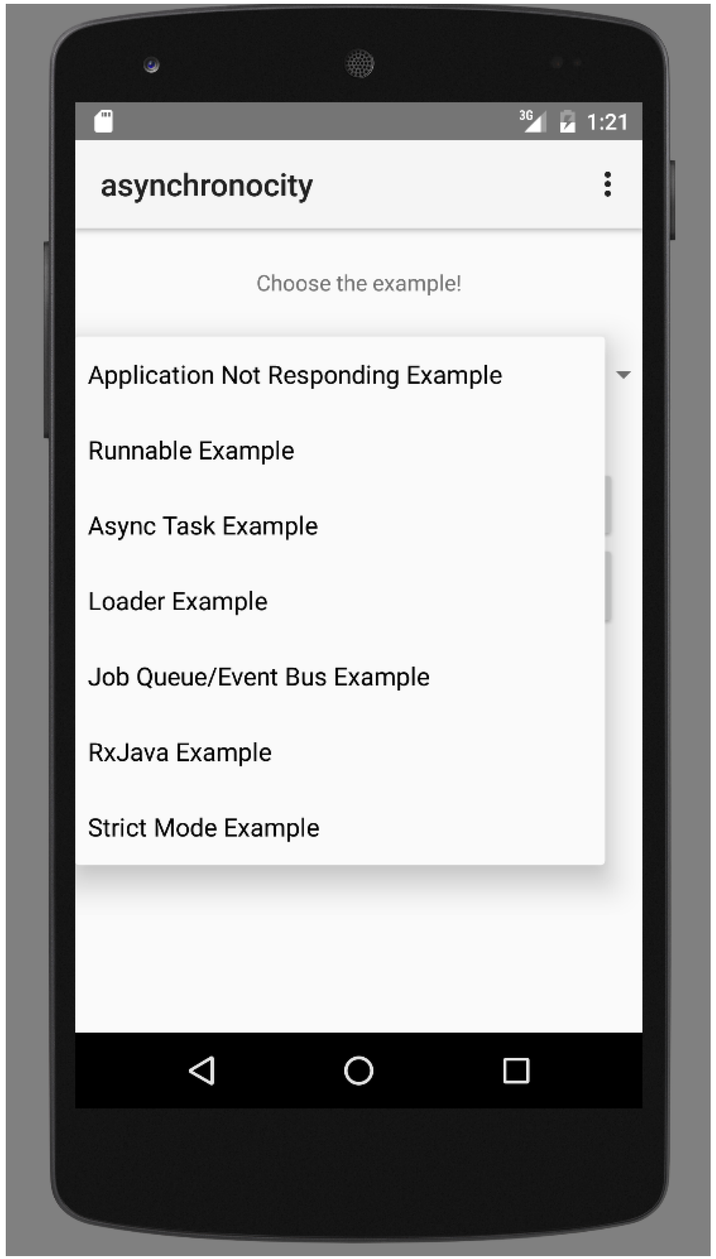
Consultant at Manifest Solutions

Android developer since Froyo (v2.2)

Example Application

https://github.com/pendext/asynchronicity


1:20
asynchronocity
Choose the example!
Application Not Responding Example
Press for Application Not Responding
PRESS ME!
CLICK THIS BUTTON TO SEE ONGOING WORK ON THE UI THREAD.

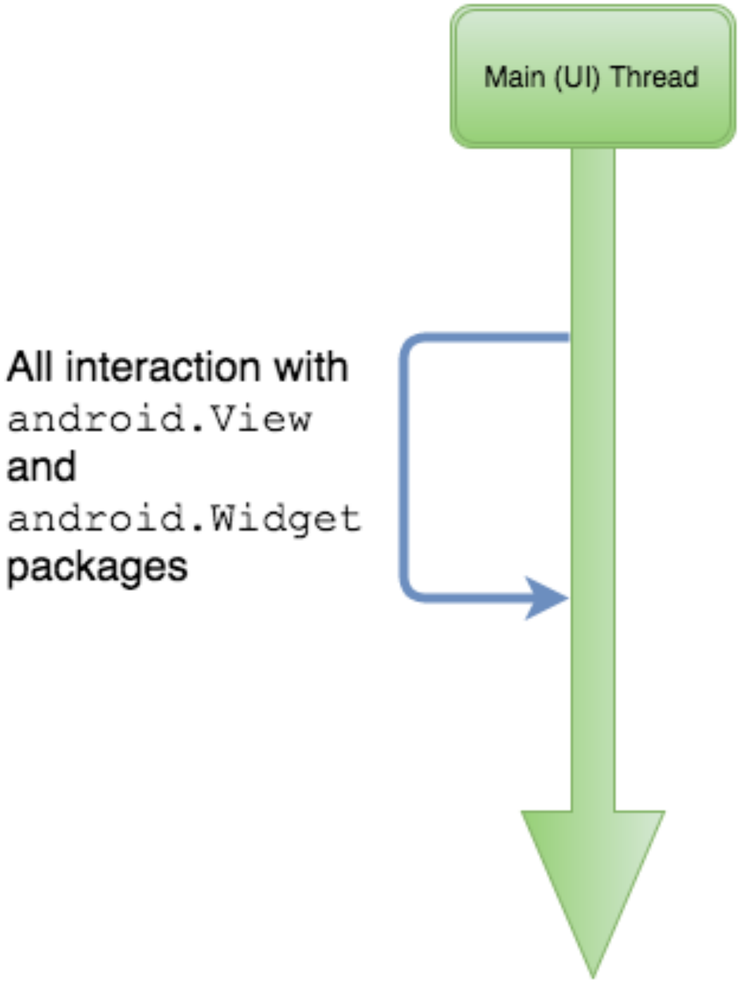

1:21
asynchronocity
Choose the example!
Application Not Responding Example
Runnable Example
Async Task Example
Loader Example
Job Queue/Event Bus Example
RxJava Example
Strict Mode Example

What does asynchronous mean in the context of Android?

UI Thread?

Main Thread?

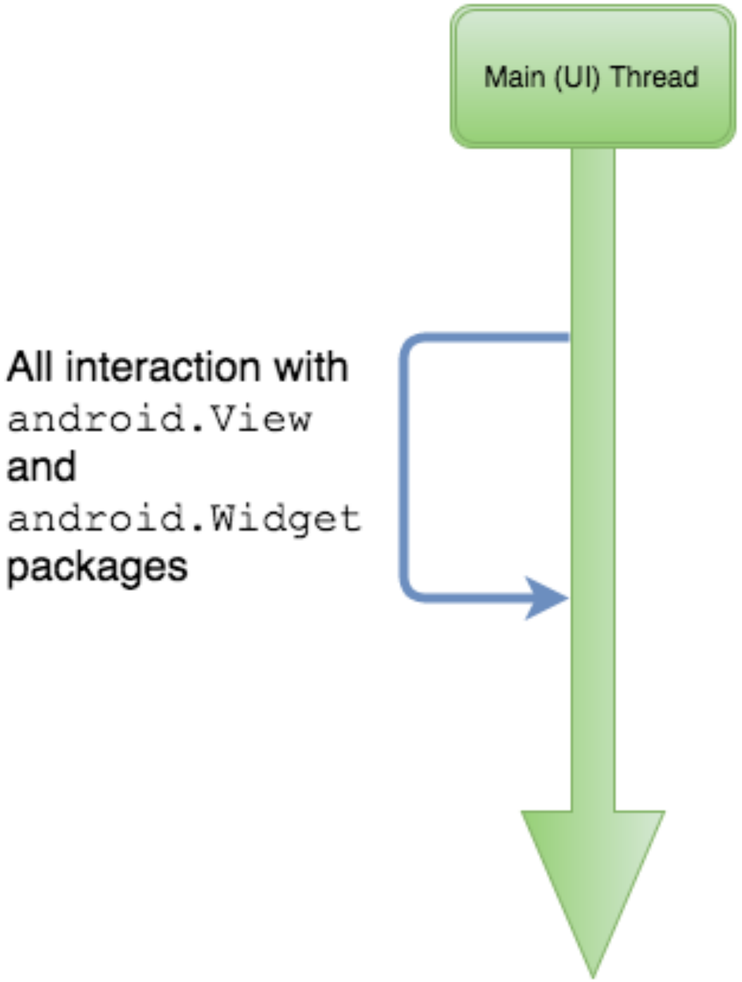
Android Threading



Android creates a thread called “main” (often referred to as the UI thread) for each application when it starts.

Source: http://developer.android.com/guide/components/processes-and-threads.html#Threads

Android Threading



Android creates a thread called “main” (often referred to as the UI thread) for each application when it starts.

- The Android operating system does not create a separate thread for each instance of a component
- Methods that respond to system callbacks (e.g. key and touch events) always run on the UI thread

Source: http://developer.android.com/guide/components/processes-and-threads.html#Threads

Android Threading Rules

The Android developer guidelines has 2 rules for dealing with threads on the Android platform

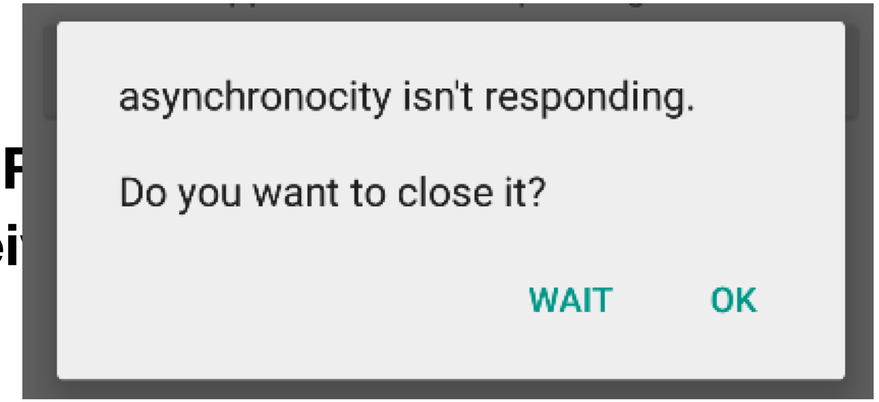
1. **Do not block the UI thread**
2. **Do not access the Android UI toolkit (e.g. `android.View` and `android.Widget`) from outside the UI thread**

Source: http://developer.android.com/guide/components/processes-and-threads.html#Threads

The dreaded ANR
(application not responding)

The Application Not Responding event occurs when an app is in a state where it cannot receive user input.

The dreaded ANR
(application not responding)



The Application Not R **an app is in a state where it cannot recei**

The dreaded ANR
(application not responding)

The Application Not Responding event occurs when an app is in a state where it cannot receive user input.

The dreaded ANR
(application not responding)

The Application Not Responding event occurs when an app is in a state where it cannot receive user input.

Application Not Responding criteria

- **No response to an input event (such as key press or screen touch events) within 5 seconds**
- **A BroadcastReceiver hasn't finished executing within 10 seconds**

Source: http://developer.android.com/training/articles/perf-anr.html

Why Asynchronously?

- Application Performance
- Better User Experience
- Asynchronous code has a precedence in the world of browsers

Why Asynchronously?

- Application Performance
- Better User Experience
- Asynchronous code has a precedence in the world of browsers

```
$("#a_button").click(function() {
    $.ajax({
        url: "http://www.someRestfulEndpoint.com/",
        success: function(output) {
            $("#a_div").html(output);
    }});
});
```

What Asynchronously?

- Network calls
- Queries against a local SQLite database
- Anything computationally intensive
- Background tasks/services/jobs

Asynchronous in the Android SDK

Thread & Runnable.run()

- Very low level
- Does not have access to the UI thread, requiring usage of one of the following methods of accessing the UI thread

```
Activity.runOnUiThread(Runnable runnable)

View.post(Runnable runnable)

View.postDelayed(Runnable runnable, long delay)

Handler
```

Basic Runnable.run() Usage

```
public class Service() {

    Activity activity;
    TextView textView;
    RestService restService;

    public Service(Activity activity, TextView textView) {
        this.activity = activity;
        this.textView = textView;
        this.restService = new RestService();
    }

    public void longRunningEvent() {
        // Any long running event could go here
        String data = restService.getData();
        activity.runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    textView.setText(data);
                }
        });
    }
}
```

Basic Runnable.run() Usage

Basic Runnable.run() Usage

```
public class TextRunnable implements Runnable {

    private Service Service;

    public TextRunnable(Service Service) {
        this.service = service;
    }

    @Override
    public void run() {
        service.longRunningEvent();
    }
}
```

Basic Runnable.run() Usage

Basic Runnable.run() Usage

```
new Thread(
    new TextRunnable(
        new Service(activity, textView)).start();
```

Downsides of `Runnable`

- Can be complex
- Tightly couples the long running event to an `Activity` or `Fragment`
- Unnecessary in most use cases

android.os.AsyncTask

- **`AsyncTask` is recommended for short running operations, i.e. seconds not minutes**
- **`AsyncTask` is an abstract class that has three generic type parameters, `Params`, `Progress`, and `Result`**
- **`AsyncTask` excels for RESTful calls, SQLite writes, shorter one off tasks**

`AsyncTask` methods/callbacks

`onPreExecute()` - Invoked on the UI thread before the task is executed. Any setup should go here

`doInBackground(Params...)` - Invoked off of the UI thread. The work should go here

`onProgressUpdate(Progress...)` - Invoked on the UI thread. Any display of progress to the user should go here

`onPostExecute(Result)` - Invoked on the UI thread after `doInBackground()` returns. The result of `doInBackground()` is passed into this method as a parameter

`AsyncTask` **usage**

From within an `Activity` or `Fragment`

```
new BasicAsyncTask().execute(parameters);
```

- **`execute()` must be invoked on the UI thread**
- **Do not call any of the previous methods/callbacks directly**
- **The task can be executed only once**

Downsides of `AsyncTask`

- Tightly couples the asynchronous work to an `Activity` or `Fragment` and the activity life cycle
- Should only be used for shorter asynchronous work
- Canceling an `AsyncTask` still completes the work in `doInBackground()`, forcing the invoker of the AsyncTask to handle canceled requests that actually complete

android.content.Loader

- **Loaders are available to any `Activity` or `Fragment`**
- **Loaders monitor the source of the data and and update the results when the data changes**
- **Loaders are good for accessing changing data within a SQLite database or from within a Content Provider**

android.content.Loader

Important classes:

- **`LoaderManager` - managers loaders in the context of the Android Activity Lifecycle**
- **`LoaderManager.LoaderCallbacks` - Interface that must be implemented, contains methods that allow the UI to be updated when the data changes**

```
public class LoaderFragment extends Fragment implements LoaderManager.LoaderCallbacks<Cursor> {

    private SimpleCursorAdapter adapter;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        ...

        getLoaderManager().initLoader(0, null, this);

        CommentDao commentDao = new CommentDao(getActivity().getApplicationContext());

        String[] from = new String[] { ASqliteOpenHelper.COMMENT_COLUMN };
        int[] to = new int[] { R.id.loader_label };
        adapter = new SimpleCursorAdapter(getActivity().getApplicationContext(),
                R.layout.layout_for_loader, commentDao.getCommentCursor(), from, to);

        commentListView.setAdapter(adapter);

        return rootView;
    }
```

```
@Override
public Loader<Cursor> onCreateLoader(int id, Bundle args) {
    return new CursorLoader(getActivity().getApplicationContext(),
            Uri.parse(CommentContentProvider.URI), null,
            null, null,
            null);
}
```

```
@Override
public void onLoadFinished(Loader<Cursor> loader, Cursor data) {
    adapter.changeCursor(data);
}
```

```
@Override
public void onLoaderReset(Loader<Cursor> loader) {
    adapter.changeCursor(null);
}
```

android.app.Service

- **Services in Android function very much like an `Activity` without a UI component, including a lifecycle similar to the Android Activity lifecycle**
- **Services do not have to be bound to an `Activity`**
- **Services will run indefinitely even if the component they are called from is destroyed**
- ***Services run on whichever thread they are called from so to run off of the UI thread, they must be invoked from a separate `Thread`, or should do any long running work in the service on a separate `Thread`***

android.app.Service

Services are good for tasks that will be continuously running on a device and updating the user or completing some task regardless of the state of the application

e.g.

A cloud based file storage application that syncs new photos automatically to the cloud as they are taken

Usage is as simple as calling `startService()` from within an `Activity` and passing this method an Intent with the desired `Service`

Open Source Asynchronicity

Square

green robot

Priority Job Queue

https://github.com/yigit/android-priority-jobqueue

The priority job queue provides a framework to define tasks that run off of the UI thread

- **Prioritization**
- **Persistance**
- **Delaying**
- **Network control**

Priority Job Queue Usage

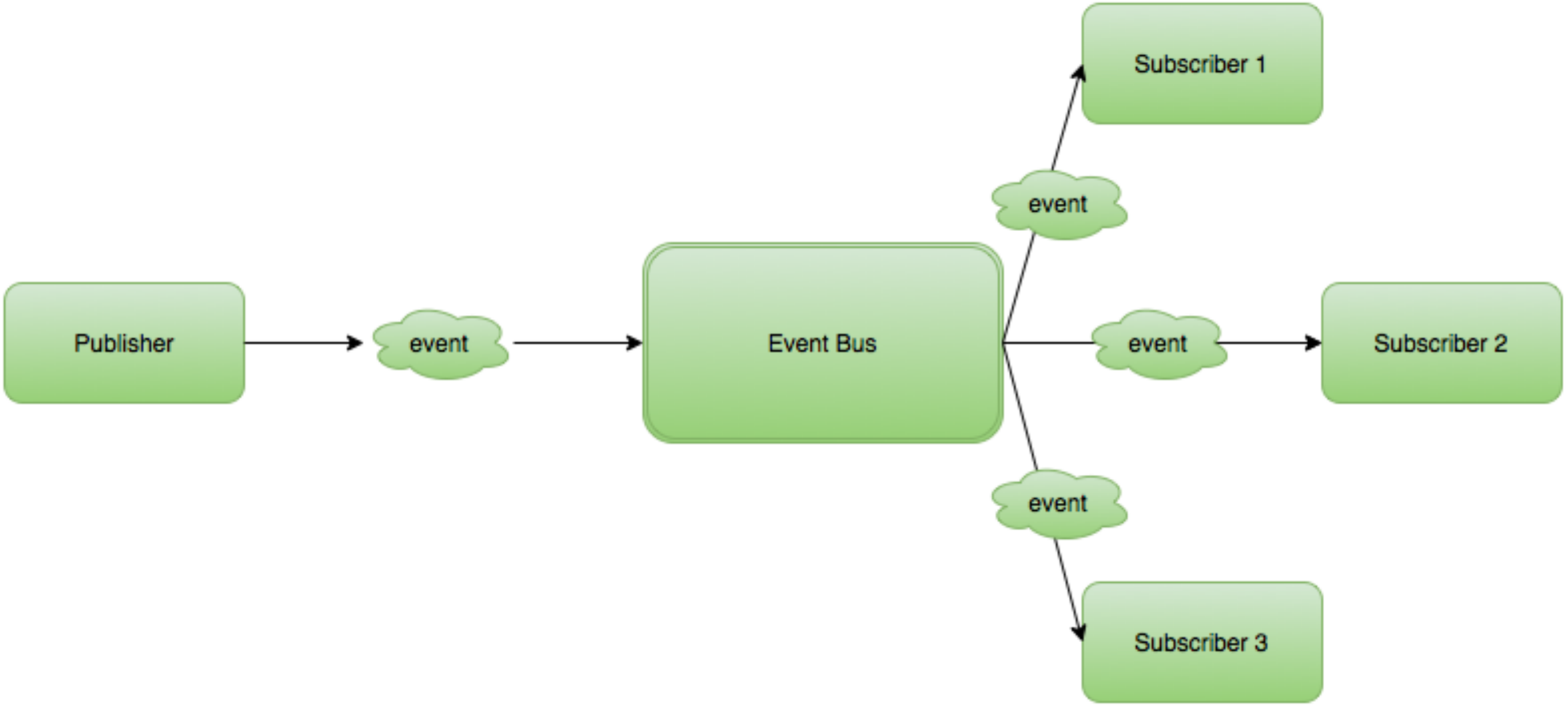
```
public class AsynchronousJob extends Job {

    public static final int PRIORITY = 1;

    public AsynchronousJob() {
        super(new Params(PRIORITY).requireNetwork().persist());
    }

    @Override
    public void onAdded() { ... }

    @Override
    public void onRun() throws Throwable { ... }

    @Override
    protected boolean shouldReRunOnThrowable(Throwable throwable) { ... }

    @Override
    protected void onCancel() { ... }
}
```

Priority Job Queue Usage

From with an `Activity` or `Fragment`

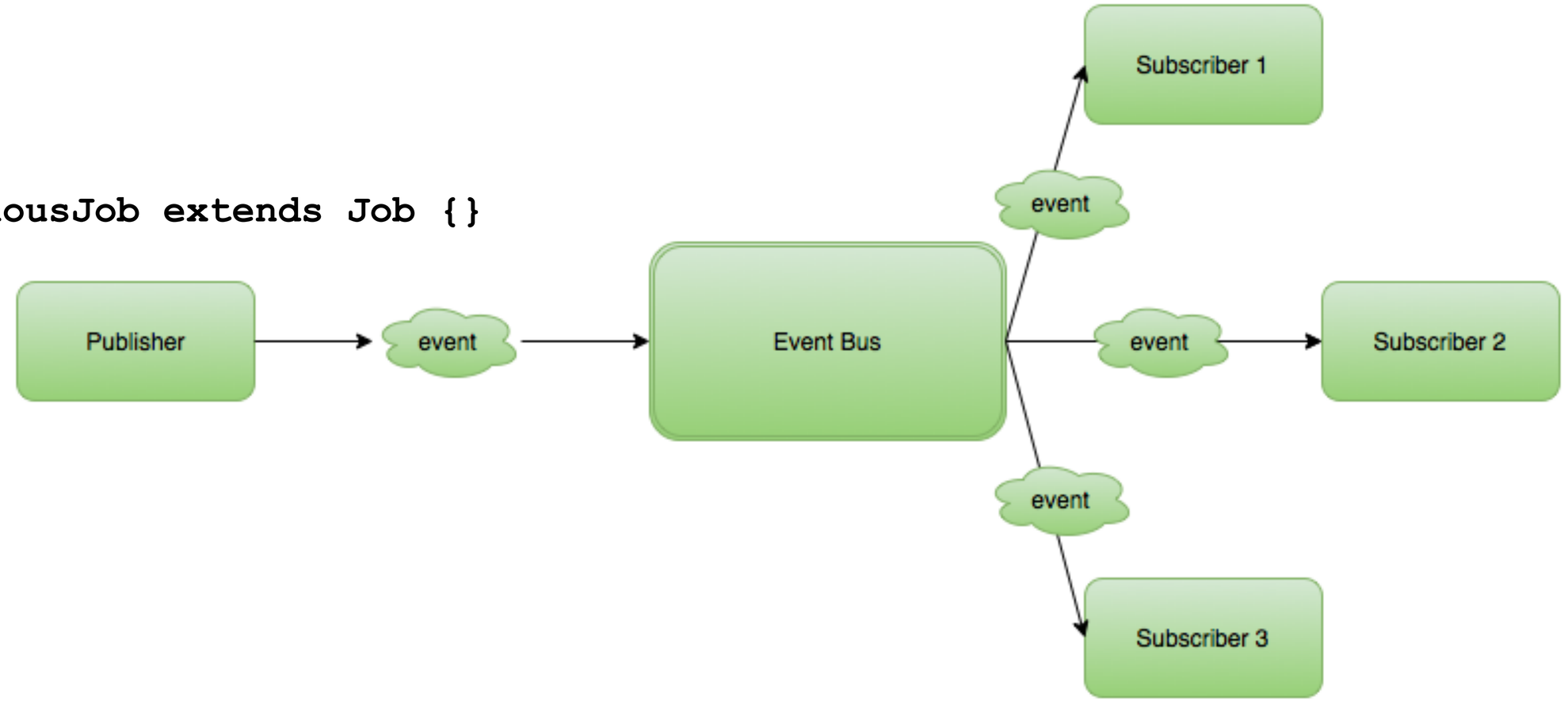
```
jobManager.addJobInBackground(new AsynchronousJob());
```

Event Bus


Publisher
event
Event Bus
Subscriber 1
event
event
Subscriber 2
event
Subscriber 3

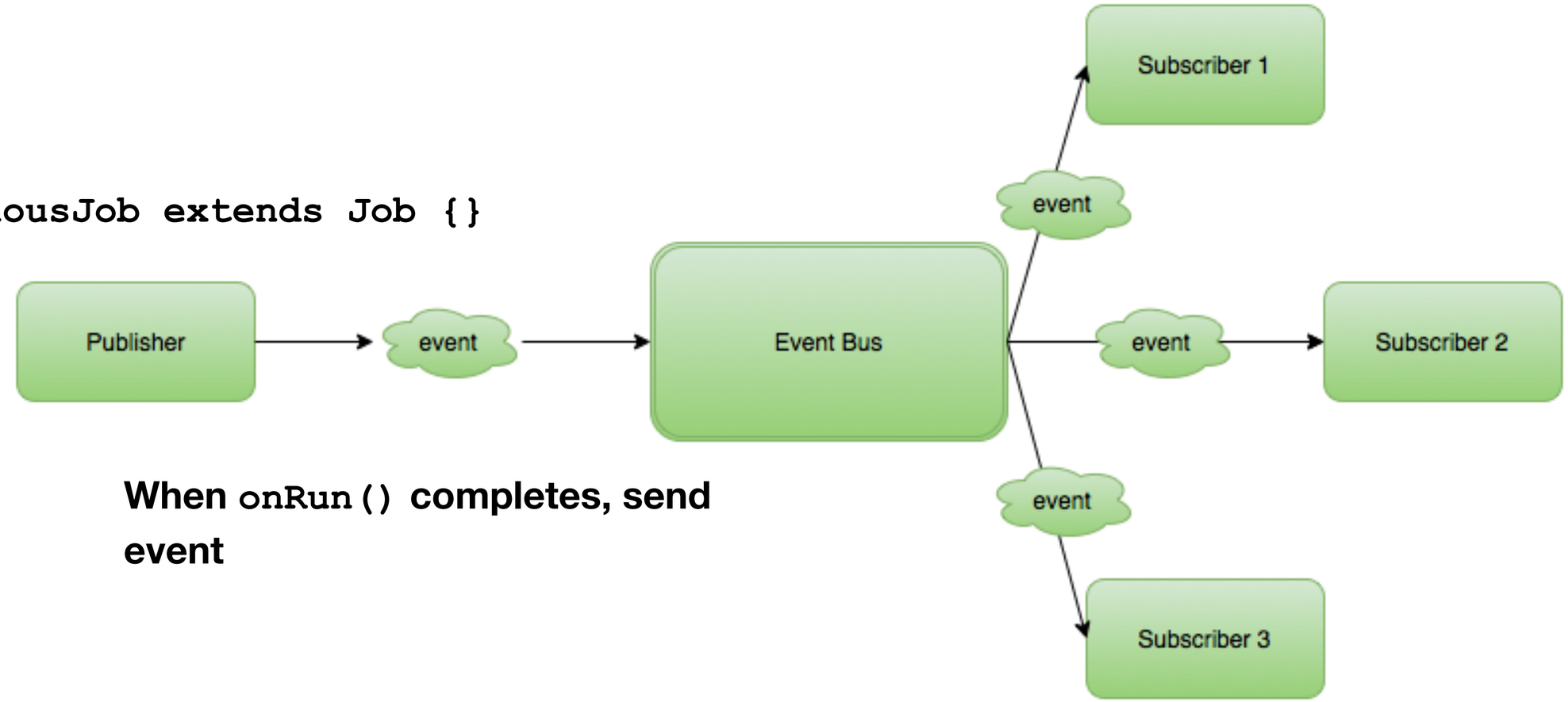
Event Bus

```
AsynchronousJob extends Job {}
```



Event Bus

```
AsynchronousJob extends Job {}
```

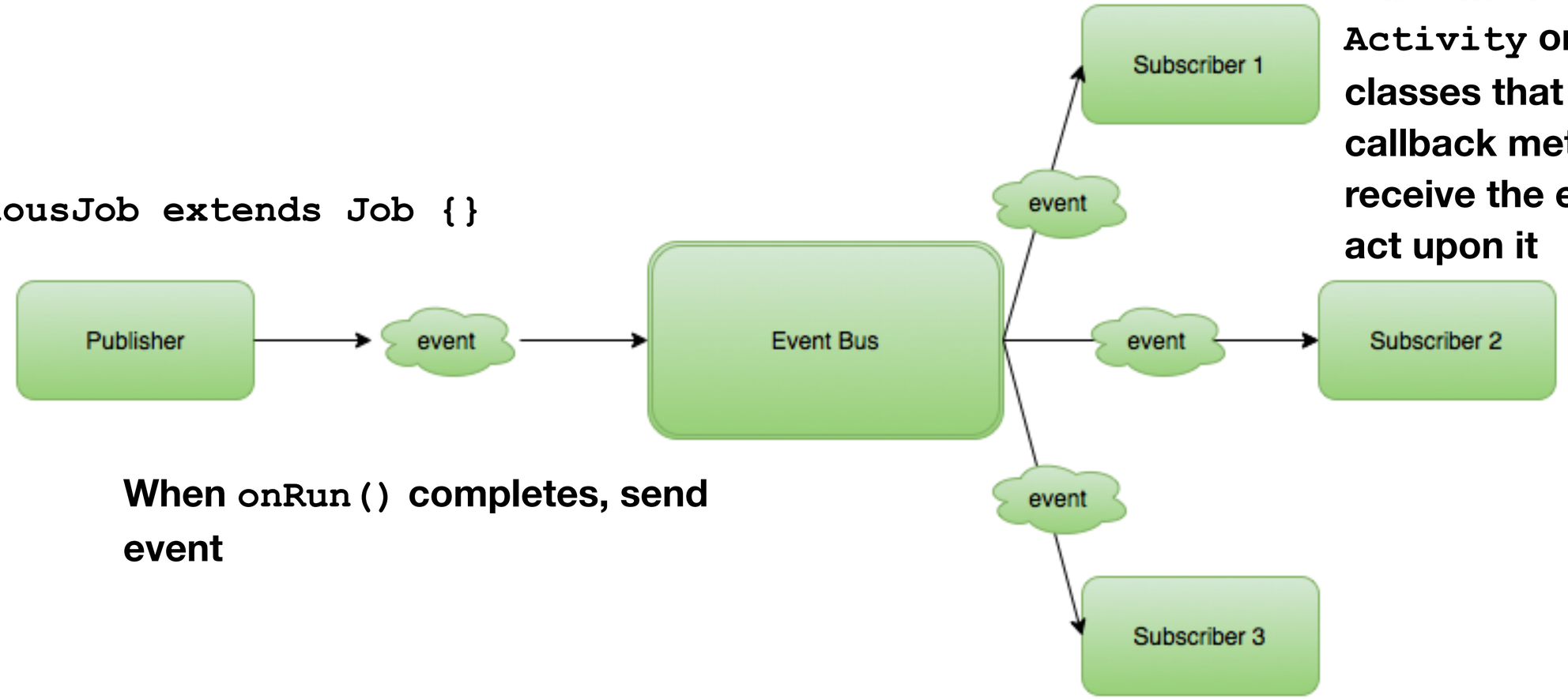


When `onRun()` completes, send event

Event Bus

Subscribers are usually `Activity` or `Fragment` classes that utilize a callback method to receive the event and act upon it

```
AsynchronousJob extends Job {}
```



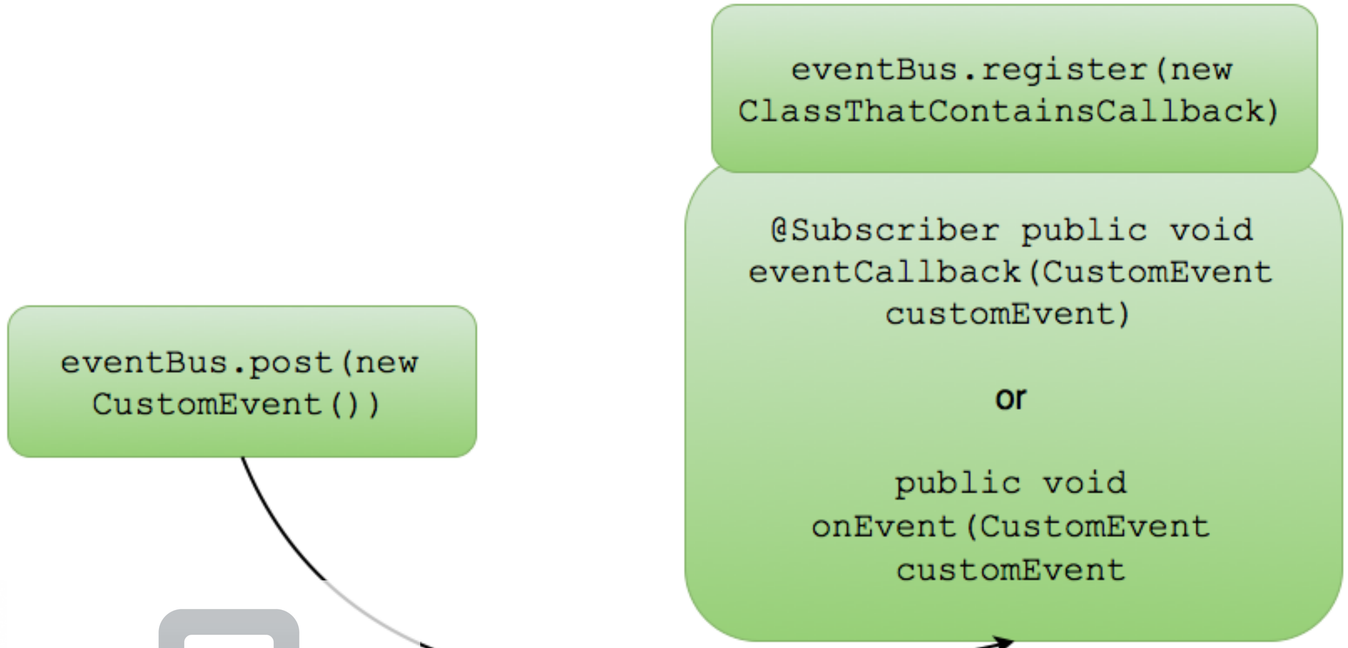
When `onRun()` completes, send event

Event Bus Implementations

https://github.com/greenrobot/EventBus

http://square.github.io/otto/

Both implementations work similarly - they provide a mechanism for sending events and a callback mechanism



green robot

Square

RxJava

ReactiveX is a library for composing asynchronous and event-based programs by using observable sequences

```
Observable<T> getData();

onNext(T);
onError(Exception);
onCompleted();
```

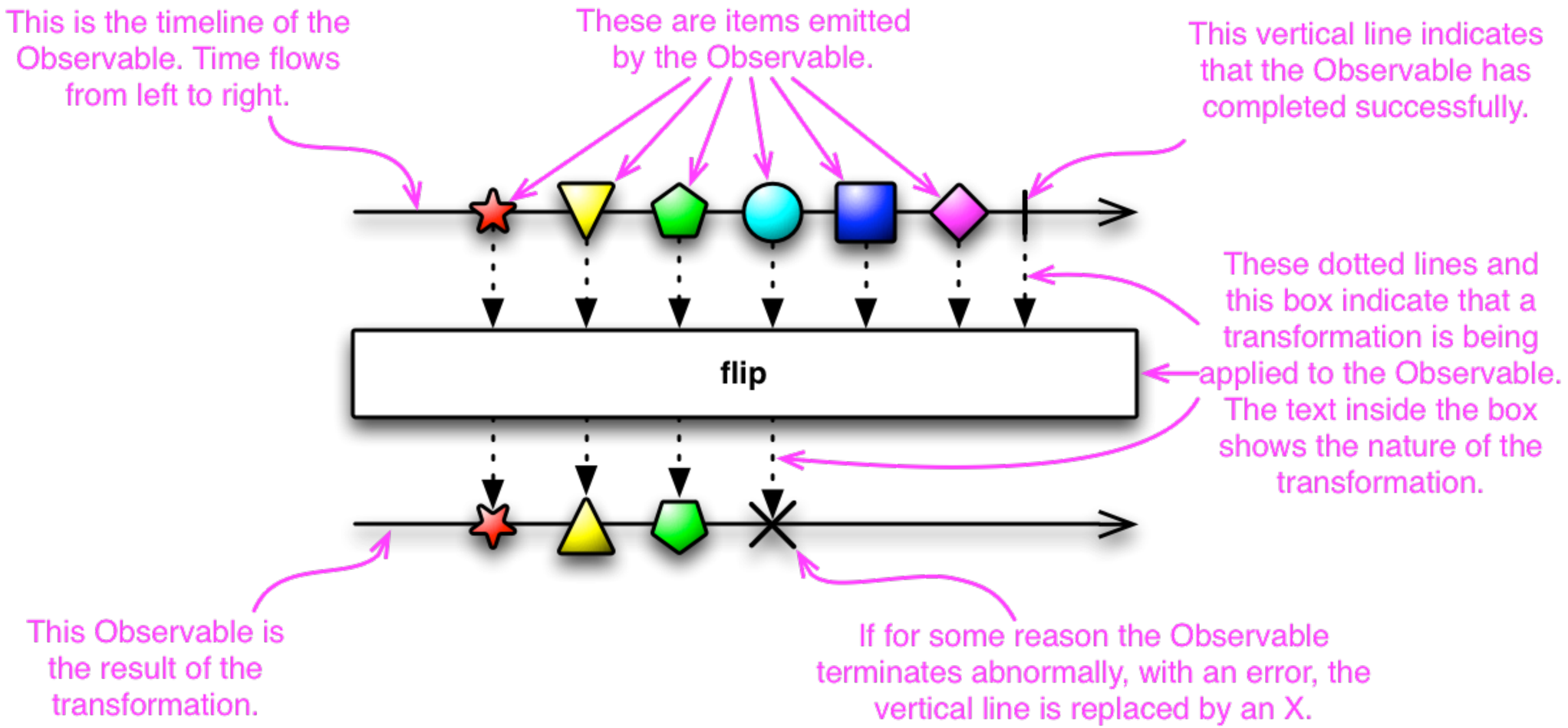
Source: http://reactivex.io/

Source: http://reactivex.io/documentation/observable.html

What does that mean?

Source: http://reactivex.io/documentation/observable.html

Source: http://reactivex.io/documentation/observable.html



Source: http://reactivex.io/documentation/observable.html

```
@Override
public void onClick(View v) {
    String invokedTime = new LocalTime().toString("hh:mm:ss");
    textView.setText("This is the initial button press at " + invokedTime);
    getData()
            .subscribeOn(Schedulers.io())
            .observeOn(AndroidSchedulers.mainThread())
            .subscribe(new Observer<String>() {
                @Override
                public void onCompleted() {}

                @Override
                public void onError(Throwable e) {}

                @Override
                public void onNext(String s) {
                    textView.setText("The long running operation ended at " + s);
                }
            });
}
```

```
public Observable<String> getData() {
    return Observable.just("").map(new Func1<String, String>() {
        @Override
        public String call(String aString) {
            String invokedTime = null;
            try {
                Thread.sleep(5000);
                invokedTime = new LocalTime().toString("hh:mm:ss");
            } catch (InterruptedException e) {}
            return invokedTime;
        }
    });
}
```

RxJava Android Module

Provides Android specific bindings for RxJava, which include wrappers around Android's `Handler` class as well as operators that are specific to the android `Activity` and `Fragment` lifecycle

Source: https://github.com/ReactiveX/RxJava/wiki/The-RxJava-Android-Module

Monitoring the UI Thread

Easy UI Thread Monitoring

Developer Options on device - Strict mode enabled

- **Device wide**
- **Minimal amount of control**

Strict mode in your application itself, programmatically

- **At the Activity or Fragment level**
- **Large amount of control**
- **Varying levels of effects when rules are violated**

Easy UI Thread Monitoring

```
StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
    .detectAll()
    .penaltyLog()
    .build();
StrictMode.setThreadPolicy(policy);
```

```
StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
    .detectAll()
    .penaltyFlashScreen()
    .build();
StrictMode.setThreadPolicy(policy);
```

Easy UI Thread Monitoring

```
StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
    .detectAll()
    .penaltyDialog()
    .build();
StrictMode.setThreadPolicy(policy);
```

```
StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder()
    .detectAll()
    .penaltyDeathOnNetwork()
    .build();
StrictMode.setThreadPolicy(policy);
```

If nothing else remember…

1. **Do not block the UI thread**
2. **Do not access the Android UI toolkit (e.g. `android.View` and `android.Widget`) from outside the UI thread**

Source: http://developer.android.com/guide/components/processes-and-threads.html#Threads

Questions
?

jpendexter@manifestcorp.com