Android Developer Fundamentals

Background Tasks

Lesson 7

7.1 AsyncTask & AsyncTaskLoader

Contents

- Threads
- AsyncTask
- Loaders
- AsyncTaskLoader

Threads

The main thread

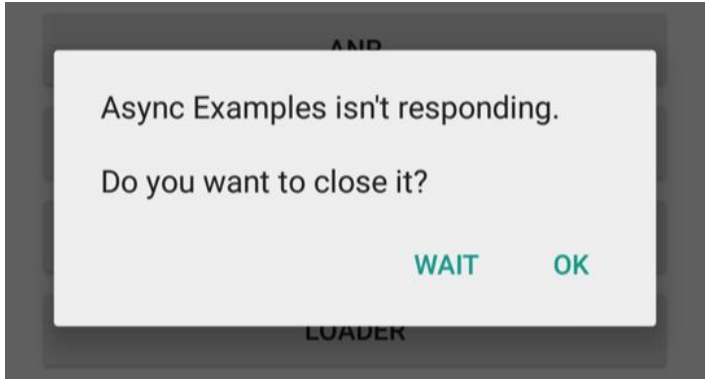
- Independent path of execution in a running program
- Code is executed line by line
- App runs on Java thread called "main" or "UI thread"
- Draws UI on the screen
- Responds to user actions by handling UI events

The Main thread must be fast

- Hardware updates screen every 16 milliseconds
- UI thread has 16 ms to do all its work
- If it takes too long, app stutters or hangs

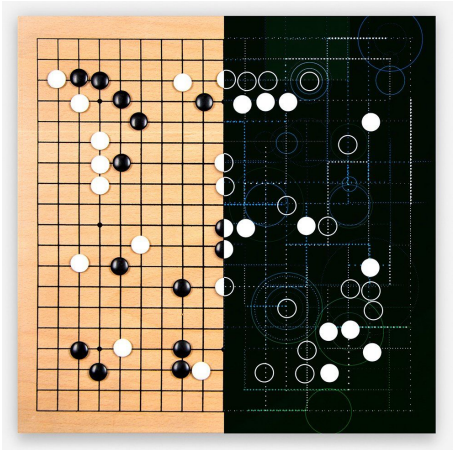
Users uninstall unresponsive apps

- If the UI waits too long for an operation to finish, it becomes unresponsive
- The framework shows an Application Not Responding (ANR) dialog



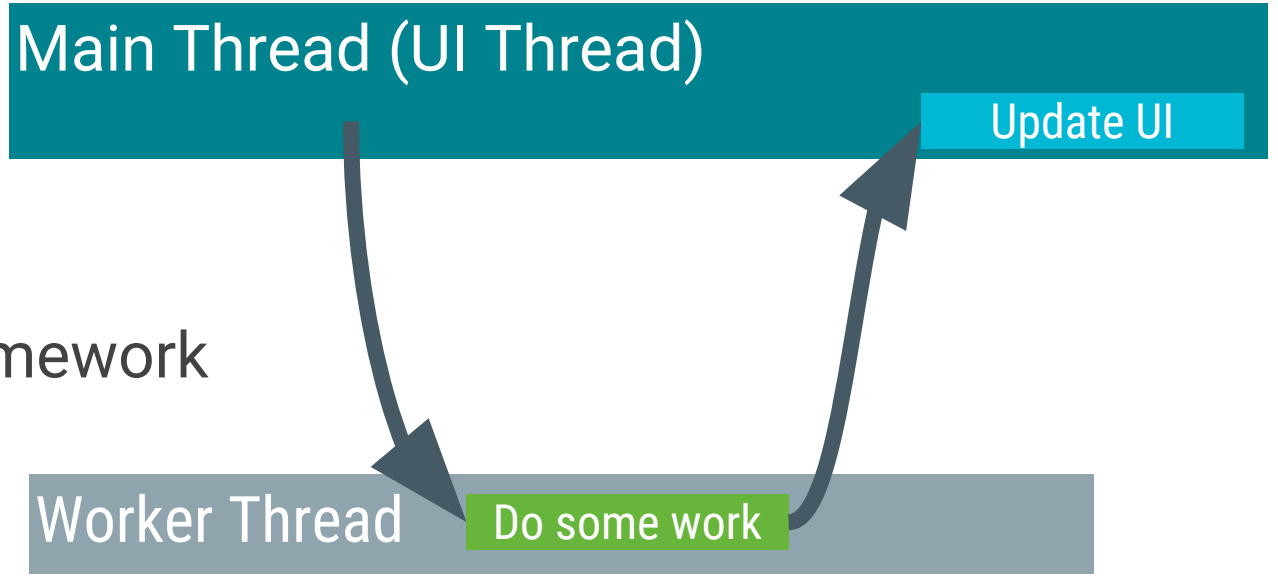
What is a long running task?

- Network operations
- Long calculations
- Downloading/uploading files
- Processing images
- Loading data



Background threads

Execute long running tasks on a **background thread**



- AsyncTask
- The Loader Framework
- Services

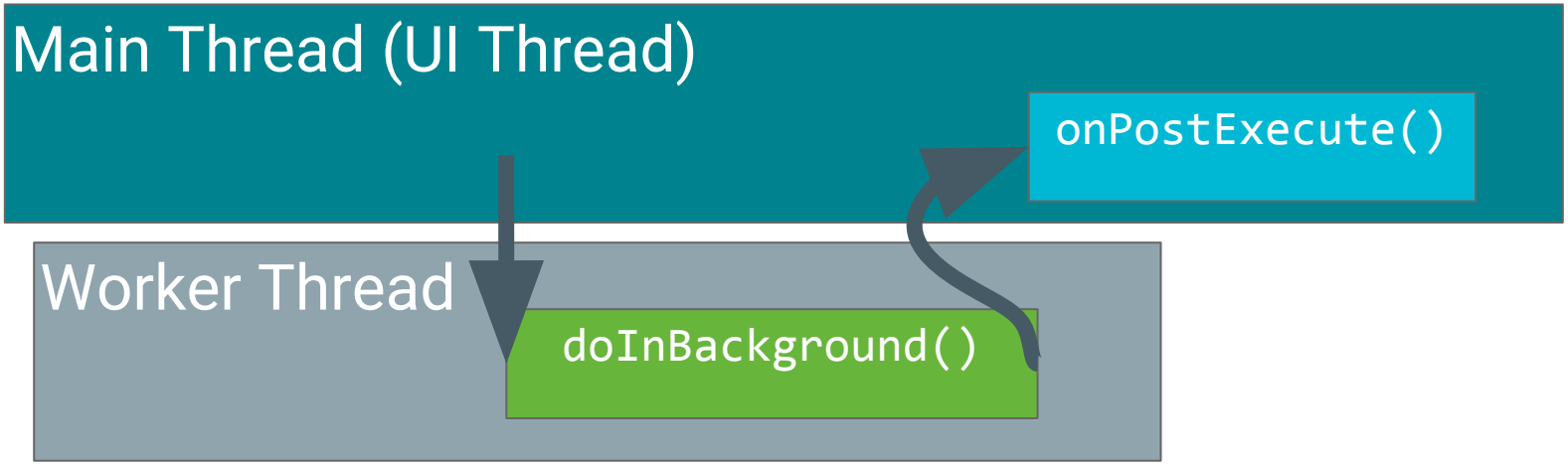
Two rules for Android threads

- Do not block the UI thread
  - Complete all work in less than 16 ms for each screen
  - Run slow non-UI work on a non-UI thread
- Do not access the Android UI toolkit from outside the UI thread
  - Do UI work only on the UI thread

AsyncTask

What is AsyncTask?

Use AsyncTask to implement basic background tasks



Override two methods

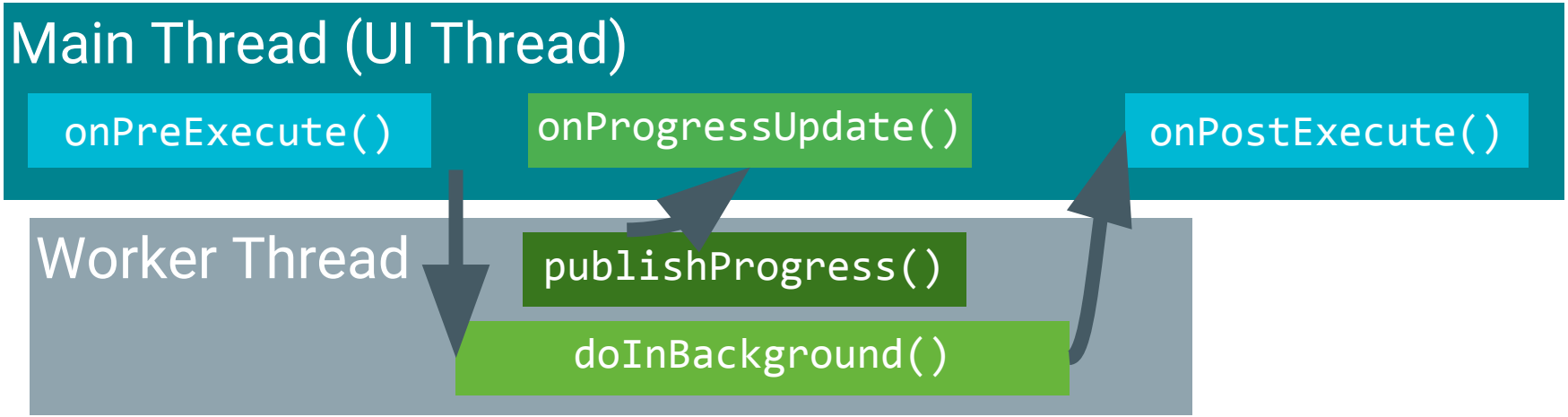
- `doInBackground()`—runs on a background thread
  - All the work to happen in the background
- `onPostExecute()`—runs on main thread when work done
  - Process results
  - Publish results to the UI

AsyncTask helper methods

- `onPreExecute()`
  - Runs on the main thread
  - Sets up the task

- `onProgressUpdate()`
  - Runs on the main thread
  - receives calls from `publishProgress()` from background thread

AsyncTask helper methods



Creating an AsyncTask

1. Subclass AsyncTask
2. Provide data type sent to `doInBackground()`
3. Provide data type of progress units for `onProgressUpdate()`
4. Provide data type of result for `onPostExecute()`

```
private class MyAsyncTask
    extends AsyncTask<URL, Integer, Bitmap> {...}
```

MyAsyncTask class definition

```
private class MyAsyncTask
    extends AsyncTask<String, Integer, Bitmap> {...}
```

- String—could be query, URI for filename
- Integer—percentage completed, steps done
- Bitmap—an image to be displayed
- Use Void if no data passed

onPreExecute()

```
protected void onPreExecute() {
    // display a progress bar
    // show a toast
}
```

doInBackground()

```
protected Bitmap doInBackground(String... query) {
    // Get the bitmap
    return bitmap;
}
```

onProgressUpdate()

```
protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
```

onPostExecute()

```
protected void onPostExecute(Bitmap result) {
    // Do something with the bitmap
}
```

Start background work

```
public void loadImage (View view) {
    String query = mEditText.getText().toString();
    new MyAsyncTask(query).execute();
   }
```

Limitations of AsyncTask

- When device configuration changes, Activity is destroyed
- AsyncTask cannot connect to Activity anymore
- New AsyncTask created for every config change
- Old AsyncTasks stay around
- App may run out of memory or crash

When to use AsyncTask

- Short or interruptible tasks
- Tasks that do not need to report back to UI or user
- Lower priority tasks that can be left unfinished
- Use AsyncTaskLoader otherwise

Loaders

What is a Loader?

- Provides asynchronous loading of data
- **Reconnects to Activity after configuration change**
- Can monitor changes in data source and deliver new data
- Callbacks implemented in Activity
- Many types of loaders available
  - AsyncTaskLoader, CursorLoader

Why use loaders?

- Execute tasks OFF the UI thread
- LoaderManager handles configuration changes for you
- Efficiently implemented by the framework
- Users don't have to wait for data to load

What is a LoaderManager?

- Manages loader functions via callbacks
- Can manage multiple loaders
  - loader for database data, for AsyncTask data, for internet data…

Get a loader with initLoader()

- Creates and starts a loader, or reuses an existing one, including its data
- Use restartLoader() to clear data in existing loader

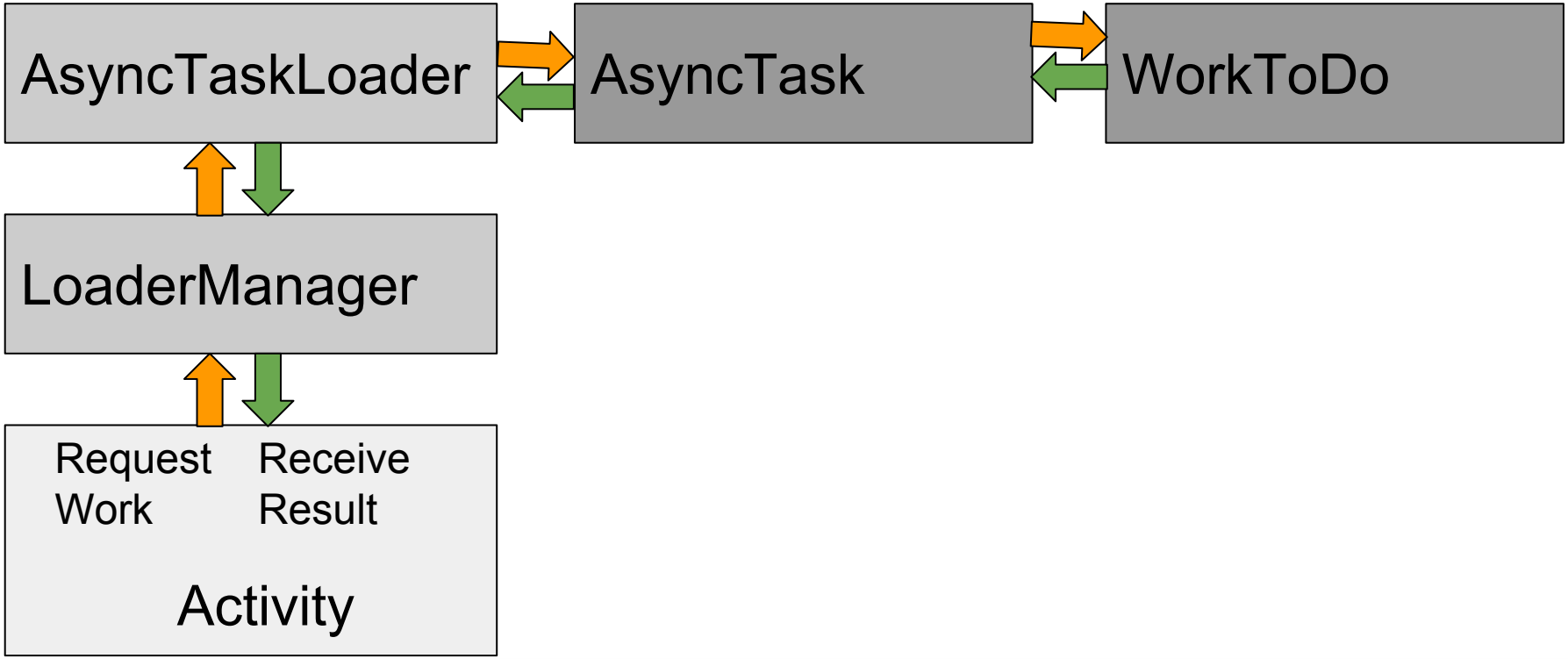
```
getLoaderManager().initLoader(Id, args, callback);

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

getSupportLoaderManager().initLoader(0, null, this);
```

Implementing AsyncTaskLoader

AsyncTaskLoader Overview



AsyncTask ⟶ AsyncTaskLoader

`doInBackground()` ⟶ `loadInBackground()`

`onPostExecute()` ⟶ `onLoadFinished()`

Steps for AsyncTaskLoader subclass

1. Subclass AsyncTaskLoader
2. Implement constructor
3. `loadInBackground()`
4. `onStartLoading()`

Subclass AsyncTaskLoader

```
public static class StringListLoader
        extends AsyncTaskLoader<List<String>> {

    public StringListLoader(Context context, String queryString) {
        super(context);
        mQueryString = queryString;
    }
}
```

loadInBackground()

```
public List<String> loadInBackground() {
    List<String> data = new ArrayList<String>;
    //TODO: Load the data from the network or from a database
    return data;
}
```

onStartLoading()

When `restartLoader()` or `initLoader()` is called, the LoaderManager invokes the `onStartLoading()` callback

- Check for cached data
- Start observing the data source (if needed)
- Call `forceLoad()` to load the data if there are changes or no cached data

```
protected void onStartLoading() {  forceLoad();  }
```

Implement loader callbacks in Activity

- `onCreateLoader()` — Create and return a new Loader for the given ID
- `onLoadFinished()` — Called when a previously created loader has finished its load
- `onLoaderReset()` — Called when a previously created loader is being reset making its data unavailable

onCreateLoader()

```
@Override
public Loader<List<String>> onCreateLoader(int id, Bundle args) {
    return new StringListLoader(this,args.getString("queryString"));
}
```

onLoadFinished()

Results of `loadInBackground()` are passed to `onLoadFinished()` where you can display them

```
public void onLoadFinished(Loader<List<String>> loader,
List<String> data) {
    mAdapter.setData(data);
}
```

onLoaderReset()

- Only called when loader is destroyed
- Leave blank most of the time

```
@Override
public void onLoaderReset(final LoaderList<String>> loader) { }
```

Get a loader with initLoader()

- In Activity
- Use support library to be compatible with more devices

```
getSupportLoaderManager().initLoader(0, null, this);
```

Learn more

- AsyncTask Reference
- AsyncTaskLoader Reference
- LoaderManager Reference
- Processes and Threads Guide
- Loaders Guide
- UI Thread Performance: Exceed the Android Speed Limit

What's Next?

- Concept Chapter: 7.1 C AsyncTask and AsyncTaskLoader
- Practical: 7.1 P Create an AsyncTask

END