CPSC475/575
Threads

Today

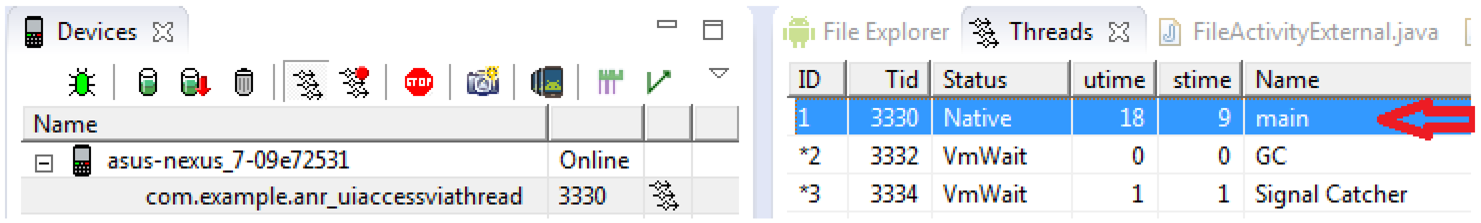
- The 2 rules
- Updating UI with Threads
- Handling Rotations

No Synchronization between Threads Yet

The 2 Rules

- **DO NOT BLOCK THE UI THREAD**
  - Long-running code in main thread will make GUI controls nonresponsive and sometimes generate an ANR.
- **ONLY THE UI THREAD CAN ACCESS UI ELEMENTS**
  - Background threads are prohibited from updating UI.

what’s the UI Thread? Its called main



Name		
asus-nexus_7-09e72531	Online	
com.example.anr_uiaccessviathread	3330	

ID	Tid	Status	utime	stime	Name
1	3330	Native	18	9	main
*2	3332	VmWait	0	0	GC
*3	3334	VmWait	1	1	Signal Catcher

Nonresponsive GUI Controls

- **Solution**
  - *Move time-consuming operations (network access, file access, database access, image manipulation or any long running task) to other threads*
  - ✓ **Threads (runnable)**– most granular, hardest to get right, useful for small tasks requiring 1 thread
  - **ExecutorService** – A framework to manage threadpools, lots of flexibility, much easier to get right
  - ✓ **AsyncTask** – Android specific wrapper around runnable
    - Very useful for task that are run off the UI thread that need to interact with UI Thread elements
    - Methods for starting and stopping, UI updating and returning a result

Threads Cannot Update UI

- Android UI toolkit is not threadsafe, you cannot update UI from other threads.
- Solutions (alternatives)
  - Wait until all threads are done, then update UI
    - When multithreading improves performance, but total wait time is small - If 1 thread then use runnable, if many use ExecutorService (not addressed here)
  - Can use thread to divide tasks between background and UI threads

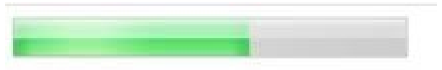
Threads in Android

AsyncTasks – the old way, recently deprecated but embedded in many code bases

Java Threads- the old and the new way, google reccomends using these

AsyncTask

Scenario

- Total wait time might be large, so you want to show intermediate results (progressbar) 
- You are designing code to divide the work between GUI and non-GUI code

Approach (4 steps)

- onPreExecute ← Runs on UI thread
- doInBackground ← Runs on Background thread
- onProgressUpdate ← Runs on UI thread
- publishProgress ← Runs on Background thread
- onPostExecute or onCancelled ← Runs on UI thread

AsyncTask: Quick Example

- **Task itself**

```
private class ImageDownloadTask extends AsyncTask<String, Void, View> {
      public View doInBackground(String... urls) {
                 //return view
       }

      public void onPostExecute(View viewToAdd) {
          //
       }
   }
```

- **Invoking task**

```
String imageAddress = "http://...";
ImageDownloadTask task = new ImageDownloadTask();
task.execute(imageAddress);
```

AsyncTask Details: Constructor

- **Class is genericized with three arguments**
  - AsyncTask<ParamType, ProgressType, ResultType>
- **Interpretation**
  - ParamType
    - This is the type you pass to execute, which in turn is the type that is send to doInBackground. Both methods use varargs, so you can send any number of params.
  - ProgressType
    - This is the type that you pass to publishProgress, which in turn is passed to onProgressUpdate (which is called in UI thread). Use Void if you do not need to display intermediate progress.
  - ResultType
    - This is the type that you should return from doInBackground, which in turn is passed to onPostExecute (which is called in UI thread).

AsyncTask Details: doInBackground

Idea

- This is the code that gets executed in the background. It **must not** update the UI.
- It takes as arguments whatever was passed to execute
- It returns a result that will be later passed to onPostExecute in the UI thread.

Code

```
private class SomeTask extends AsyncTask<Type1, Void, Type2> {
    public Type2 doInBackground(Type1... params) {
      return(doNonUiStuffWith(params));
    } ...
}
...
new SomeTask().execute(type1VarA, type1VarB);
```

The ... in the doInBackground declaration is actually part of the Java syntax, indicating varargs.

AsyncTask Details: onPostExecute

- **Idea**
  - This is the code that gets executed on the UI thread. It **can** update the UI.
  - It takes as argument whatever was returned by doInBackground
- **Code**

```
private class SomeTask extends AsyncTask<Type1, Void, Type2> {
      public Type2 doInBackground(Type1... params) {
         return(doNonUiStuffWith(params));
       }
       public void onPostExecute(Type2 result) { doUiStuff(result); }
}
...
new SomeTask(). execute(type1VarA, type1VarB);
```

AsyncTask Details: Other Methods

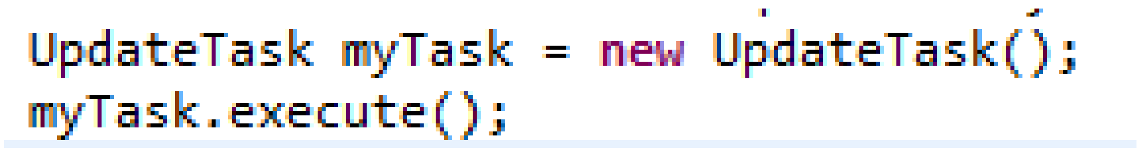
- **onPreExecute**
  - Invoked by the UI thread before doInBackground starts
- **publishProgress**
  - Sends an intermediate update value to onProgressUpdate. From background thread. You call this from code that is in doInBackground. The type is the middle value of the class declaration.
- **onProgressUpdate**
  - Invoked by the UI thread. Takes as input whatever was passed to publishProgress.
- **Note**
  - All of these methods can be omitted.

AsyncTask Details: Cancel()

- **Idea**
  - Call myAsyncTask.cancel(true);
  - Sets internal canceled flag
  - Periodically check isCanceled() in doInBackground Code
  - If canceled, onCancelled() is called verses onPostExecute

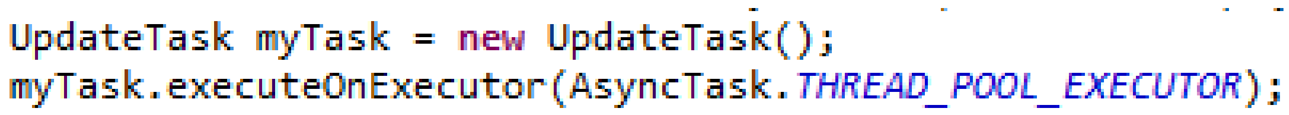
AsyncTask: problems

- Standard implementation will execute 1 AsyncTask at a time (even if you try to run many at once)



```
UpdateTask myTask = new UpdateTask();
myTask.execute();
```

- To do more than 1 at a time



```
UpdateTask myTask = new UpdateTask();
myTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
```

AsyncTask: other problems

- Deprecated as of API 30
- Solution?
- Use Java Threads, ViewModel and MutableLiveData
- See Week 8 readings

Java Threads

Relatively simple;

1) create a class that derives from Thread

```
private static class UpdateTask extends Thread {
```

2) Implement a run() method in that class

```
@Override
public void run() {
```

3) Create an instance of the thread

```
UpdateTask mt= new UpdateTask()
```

4) And start it

```
mt.start();
```

Java Threads

Relatively simple;

1) create a class that derives from Thread

```
private static class UpdateTask extends Thread {
```

2) Implement a run() method in that class

```
@Override
public void run() {
```

3) Create an instance of the thread

```
UpdateTask mt= new UpdateTask()
```

4) And start it

```
mt.start();
```

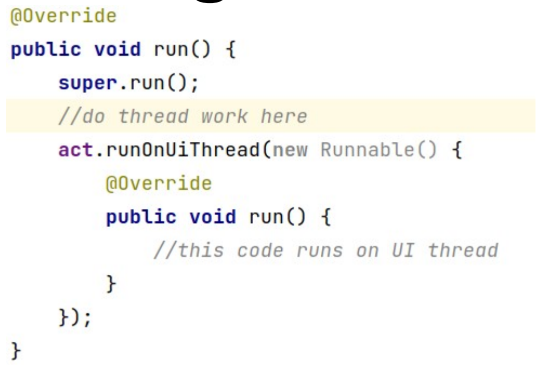
Be sure to call start() though and not run()

If you call mt.run(), UpdateTask runs in the same thread you are already in, not a new thread

Java Threads

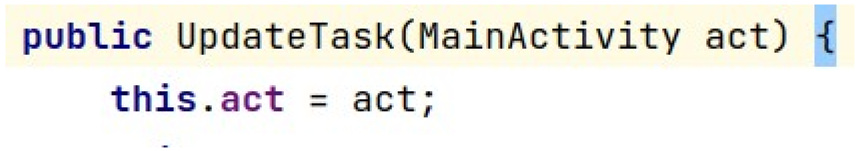
How can thread interact with UI thread?

1) Add the following to the threads run() method



```
@Override
public void run() {
    super.run();
    //do thread work here
    act.runOnUiThread(new Runnable() {
        @Override
        public void run() {
            //this code runs on UI thread
        }
    });
}
```

2) This means that the Java thread needs a ref to the activity, so add 1 in the constructor



```
public UpdateTask(MainActivity act) {
    this.act = act;
```

3) But this makes the thread 'heavily coupled' with the activity

What happens when the phone rotates?

Configuration changes

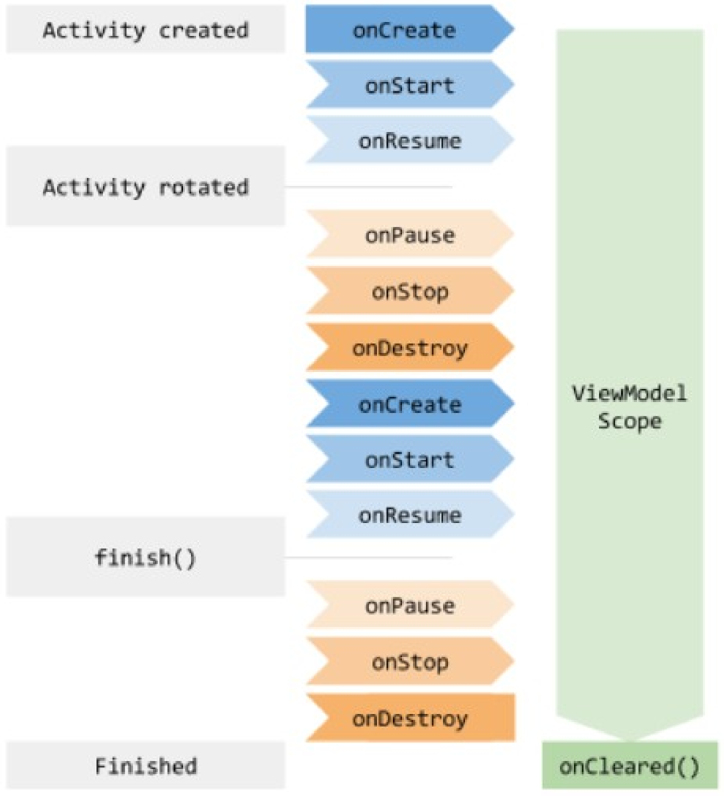
- **Problem**
  - Start an thread and then phone rotates
  - Activity is destroyed and restarted
  - Thread however is still running
  - What about all the references the Thread has to original activity?
  - Solution:
  - Use singleton to hold thread
  - In onStop() save ref to thread in singleton
  - In onStart() check to see if a thread exists in singleton, if so, recapture thread
  - see 7_Thread or 7_AsyncTask

Configuration changes

- **Problem**
  - Start an thread and then phone rotates
  - Activity is destroyed and restarted
  - Thread however is still running
  - What about all the references the Thread has to original activity?
  - Solution:
  - Or use a viewModel (androids version of a singleton)

Threads: Configuration changes- Use a ViewModel

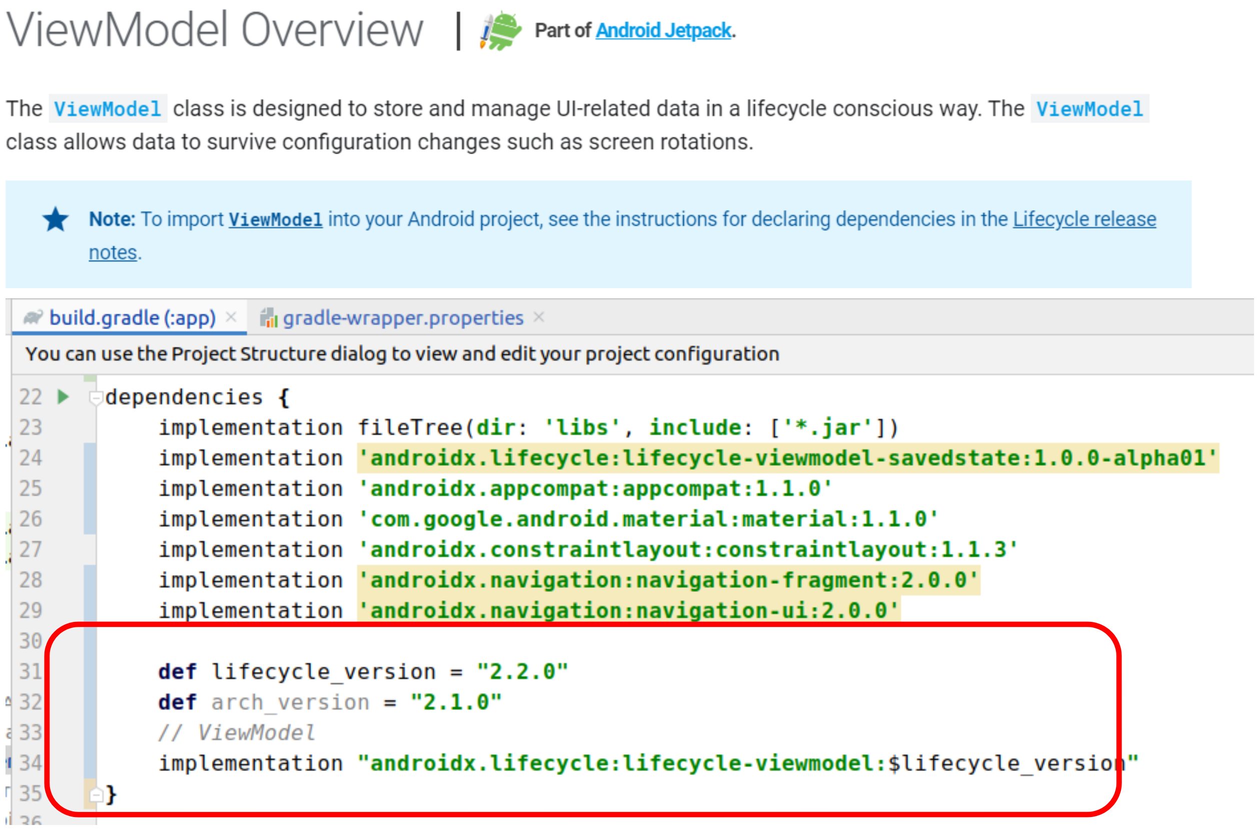
- **ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way.**



- Notice ViewModel is created in onCreate
- Persists through Activity construction/ /destruction cycles
- Is finally destroyed when app is destroyed

Threads: Configuration changes

- to use the view model you need to include some libraries in build.gradle (app) see ViewModel Overview on Course website for details.



ViewModel Overview | Part of Android Jetpack.

The `ViewModel` class is designed to store and manage UI-related data in a lifecycle conscious way. The `ViewModel` class allows data to survive configuration changes such as screen rotations.

Note: To import `ViewModel` into your Android project, see the instructions for declaring dependencies in the Lifecycle release notes.

build.gradle (:app) × gradle-wrapper.properties ×

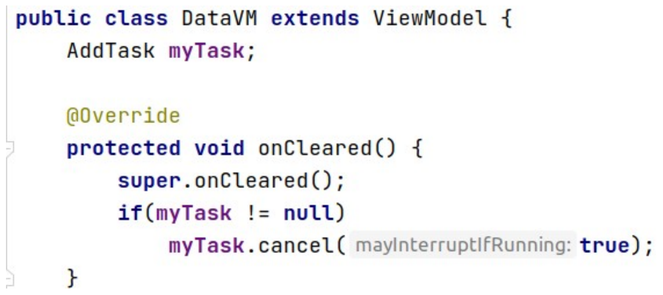
You can use the Project Structure dialog to view and edit your project configuration

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.lifecycle:lifecycle-viewmodel-savedstate:1.0.0-alpha01'
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'com.google.android.material:material:1.1.0'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    implementation 'androidx.navigation:navigation-fragment:2.0.0'
    implementation 'androidx.navigation:navigation-ui:2.0.0'

    def lifecycle_version = "2.2.0"
    def arch_version = "2.1.0"
    // ViewModel
    implementation "androidx.lifecycle:lifecycle-viewmodel:$lifecycle_version"
}
```

Threads: Configuration changes- Use a ViewModel

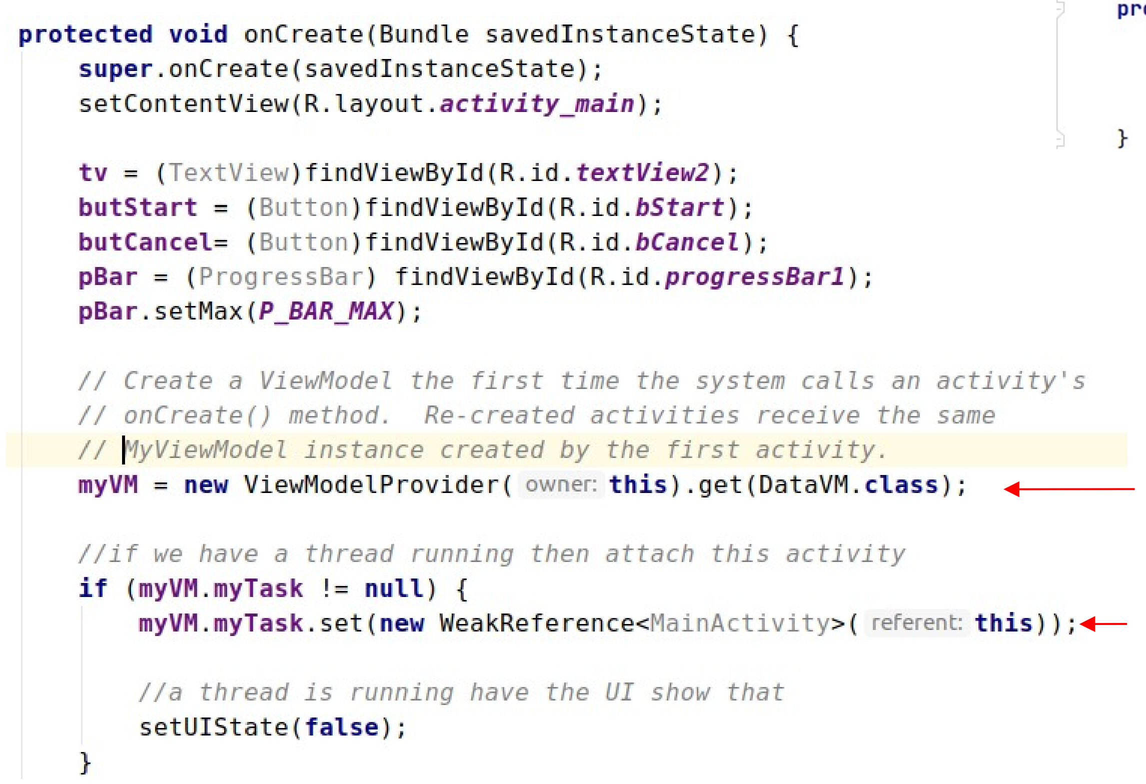
- <u>ViewModel</u> class is designed to store and manage UI-related data in a lifecycle conscious way.



```java
public class DataVM extends ViewModel {
    AddTask myTask;

    @Override
    protected void onCleared() {
        super.onCleared();
        if(myTask != null)
            myTask.cancel( mayInterruptIfRunning: true);
    }
```

Some of the ViewModel
Its Thread is a static inner class



```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    tv = (TextView)findViewById(R.id.textView2);
    butStart = (Button)findViewById(R.id.bStart);
    butCancel= (Button)findViewById(R.id.bCancel);
    pBar = (ProgressBar) findViewById(R.id.progressBar1);
    pBar.setMax(P_BAR_MAX);

    // Create a ViewModel the first time the system calls an activity's
    // onCreate() method.  Re-created activities receive the same
    // MyViewModel instance created by the first activity.
    myVM = new ViewModelProvider( owner: this).get(DataVM.class);

    //if we have a thread running then attach this activity
    if (myVM.myTask != null) {
        myVM.myTask.set(new WeakReference<MainActivity>( referent: this));

        //a thread is running have the UI show that
        setUIState(false);
    }
```

In Activity- get/create a ViewModel

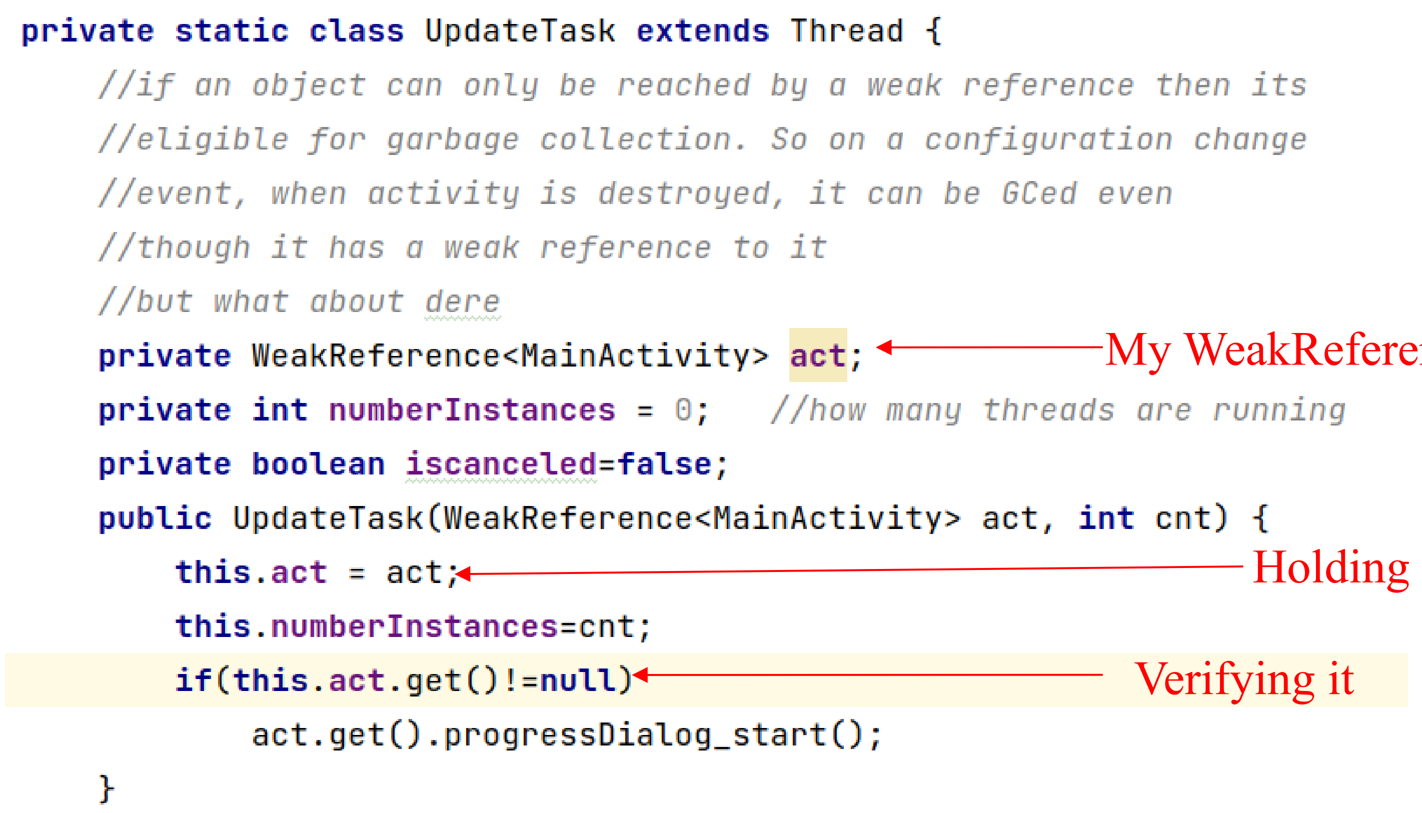
If there is a running Thread then attach it to this activity by WeakReference

Threads: Configuration changes – WeakReference?

- **Problem**: What if thread is holding a reference to an activity that has been destroyed/recreated (device rotates, phone call…)?
- If thread dereferences the destroyed Activity, you will get a null pointer exception.
- Worse, as long as thread holds this reference, Activity (and all its views and resources) cannot be Garbage Collected

- **Solution:** Hold a weak reference to the Activity!
- When activity destroyed the only ref to it will be the weakRef.
- If JVM detects an object with only weak references (i.e. no strong or soft references linked to it), this object will be marked for garbage collection.

Threads: Configuration changes – WeakReference?



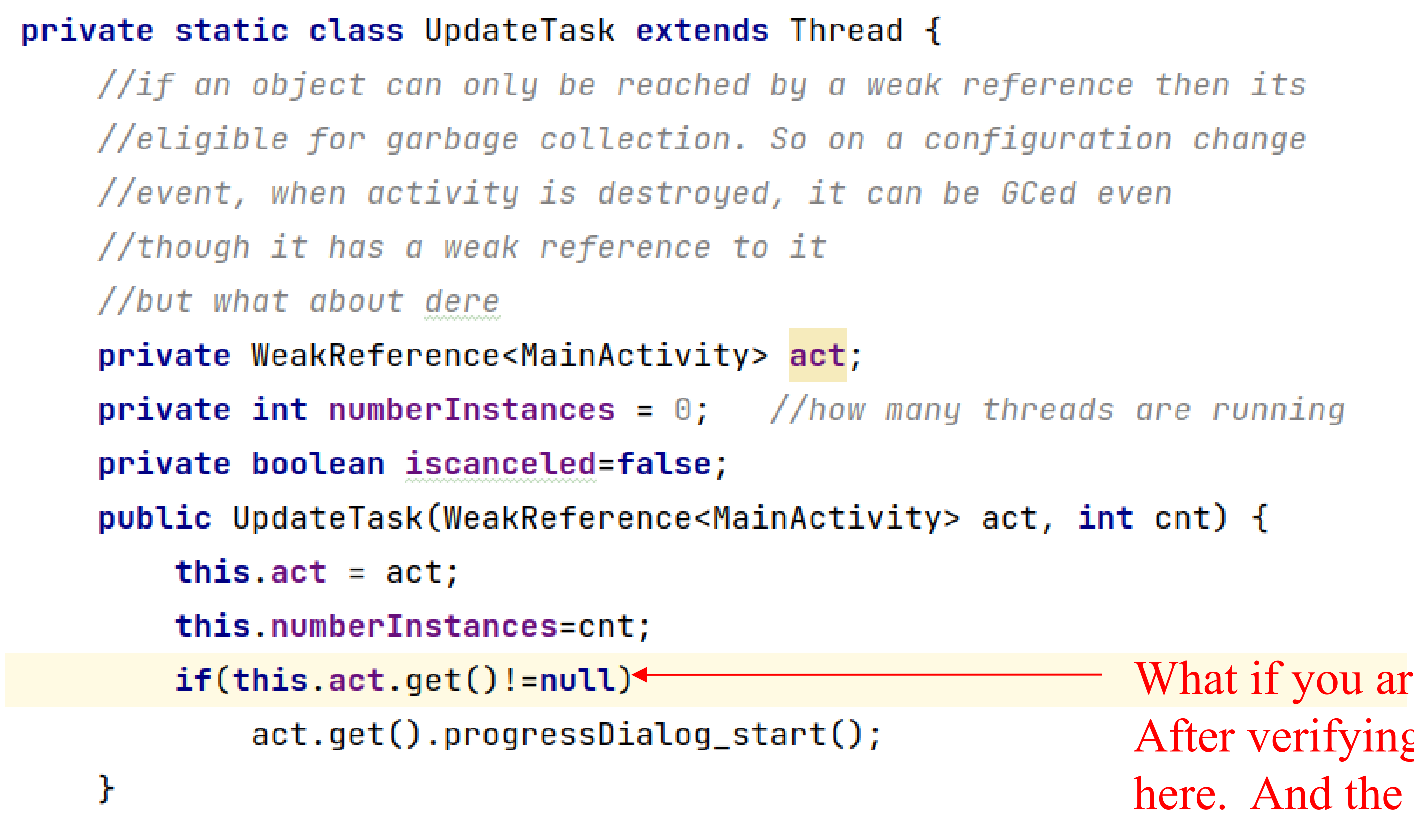
```
private static class UpdateTask extends Thread {
    //if an object can only be reached by a weak reference then its
    //eligible for garbage collection. So on a configuration change
    //event, when activity is destroyed, it can be GCed even
    //though it has a weak reference to it
    //but what about dere
    private WeakReference<MainActivity> act;
    private int numberInstances = 0;   //how many threads are running
    private boolean iscanceled=false;
    public UpdateTask(WeakReference<MainActivity> act, int cnt) {
        this.act = act;
        this.numberInstances=cnt;
        if(this.act.get()!=null)
            act.get().progressDialog_start();
    }
```

My WeakReference

Holding it

Verifying it

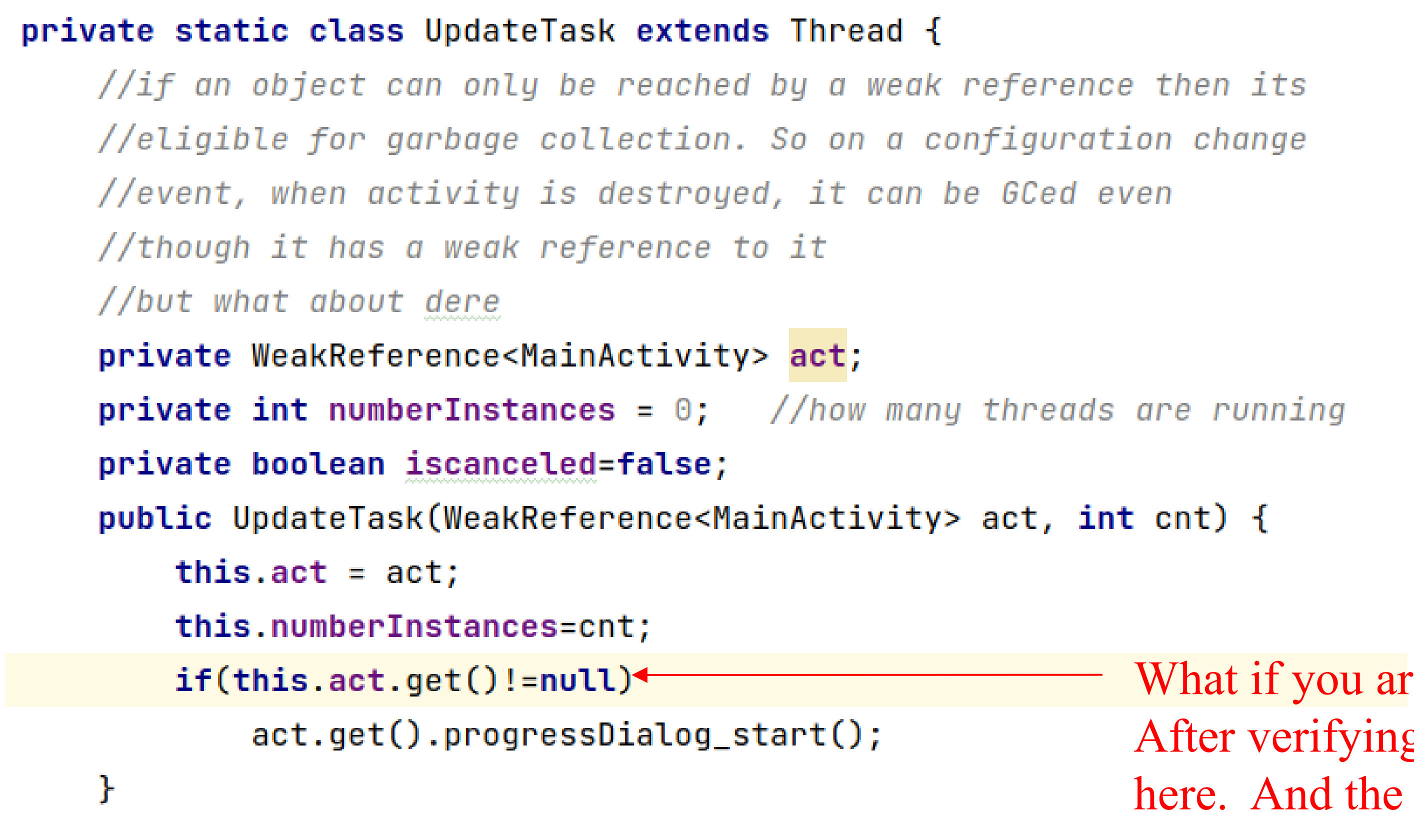
Threads: Configuration changes



```java
private static class UpdateTask extends Thread {
    //if an object can only be reached by a weak reference then its
    //eligible for garbage collection. So on a configuration change
    //event, when activity is destroyed, it can be GCed even
    //though it has a weak reference to it
    //but what about dere
    private WeakReference<MainActivity> act;
    private int numberInstances = 0;    //how many threads are running
    private boolean iscanceled=false;
    public UpdateTask(WeakReference<MainActivity> act, int cnt) {
        this.act = act;
        this.numberInstances=cnt;
        if(this.act.get()!=null)
            act.get().progressDialog_start();
    }
```

What if you are interrupted After verifying your activity here. And the activity is destroyed.

Threads: Configuration changes



```java
private static class UpdateTask extends Thread {
    //if an object can only be reached by a weak reference then its
    //eligible for garbage collection. So on a configuration change
    //event, when activity is destroyed, it can be GCed even
    //though it has a weak reference to it
    //but what about dere
    private WeakReference<MainActivity> act;
    private int numberInstances = 0;    //how many threads are running
    private boolean iscanceled=false;
    public UpdateTask(WeakReference<MainActivity> act, int cnt) {
        this.act = act;
        this.numberInstances=cnt;
        if(this.act.get()!=null)
            act.get().progressDialog_start();
    }
```

What if you are interrupted After verifying your activity here. And the activity is destroyed.

What happens when you call act.get() on next line?
Null pointer exception

Threads: Configuration changes

You either need to synchronize all global data access
Or move to something else

Google recommends mutable live data
We will talk about this next.

Reading

- **JavaDoc**
  - AsyncTask
    - http://developer.android.com/reference/android/os/AsyncTask.html
- **Tutorial: Processes and Threads**
  - http://www.javamex.com/