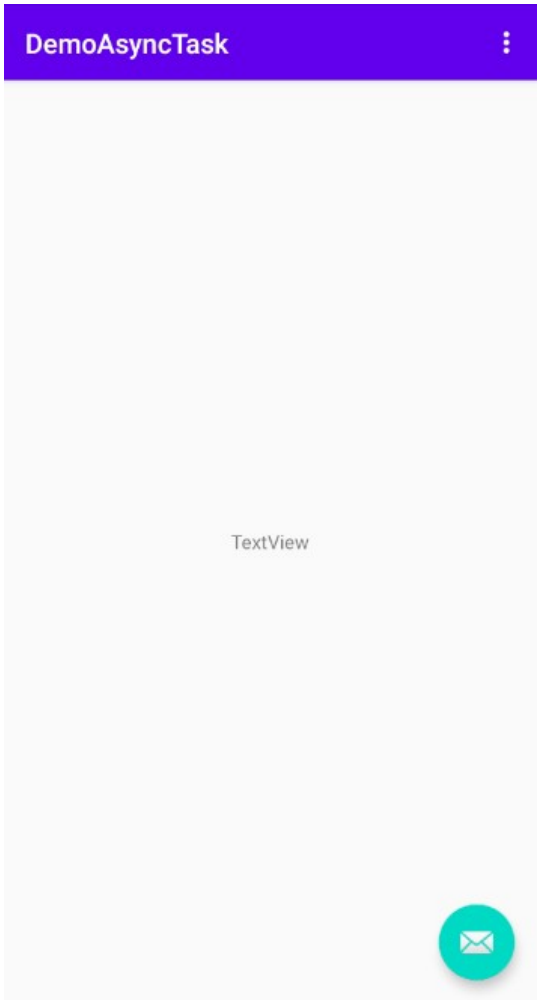
Java Thread - In class Lab and Project
I am demoing a concept so I am not going to worry about handling phone rotations
Build app as below
(I used activity with fab, and dumped the fragments.)



In activity_main.xml
the binding needs to get to content main, so give that layout an id

```
<include layout="@layout/content_main"/>
```

to

```
<include layout="@layout/content_main" android:id="@+id/main_container" />
```

In MainActivity:

have some constants

```
private static final String RUNNING_CALC = "Running Calculation on thread for ";
private static final String TIME_UNITS = " time units";
private static final String DONE = "Done with thread calculation";
private static final String USER_CANCELED = "User chose to cancel";
```

create a long running function

```
void runCalcs(Integer numb_updates)
{
    for (int i = 0; i <= numb_updates; i++) {
        try {
            Thread.sleep(ONE_SECOND);
        } catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

add associated FAB handler in OnCreate

```
binding.fab.setOnClickListener(new View.OnClickListener() {
  @Override
  public void onClick(View v) {
      //run calculation
      binding.mainContainer.textView.setText(RUNNING_CALC +
Integer.toString(numberInstances)+ TIME_UNITS); //never see this
      runCalcs(NUMBER_UPDATES);  //if this is run outside of thread never
      // see above change, button stays
      // depressed until runCalcs done
      // then see one below
      binding.mainContainer.textView.setText(DONE);

  }
});
```

Now click the do calc button (nothing happens, no text change either until everything finishes running) that's cause the main thread can't change the text until we return from this function, bummer so lets put it in a thread.

```
//this is static, so it does not hold an implicit reference
//enclosing activity, but I am explicitly holding a ref in the constructor.
Rotate the phone and activity CANNOT be Gced.  We will fix this in a bit
private static class UpdateTask extends Thread {
    private MainActivity act;
    private int numberInstances = 0;   //how many threads are running

    public UpdateTask(MainActivity act, int cnt) {
        this.act = act;
        this.numberInstances=cnt;
        act.binding.mainContainer.textView.setText(RUNNING_CALC +
Integer.toString(numberInstances)+ TIME_UNITS);
    }

    @Override
    public void run() {
        super.run();
        act.runCalcs(numberInstances);

        //act=null;    //simulates Activity being detached after above line but before next
        act.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                act.binding.mainContainer.textView.setText(DONE);
            }
        });

    }
}
```

change the FAB handler in main code

```
fab.setOnClickListener(new View.OnClickListener()
@Override
public void onClick(View v) {
UpdateTask  mt= new UpdateTask(MainActivity.this,NUMBER_UPDATES );
mt.start();

}
});
```

But all the UI is still available, and I want the user to only be able to run 1 thread at a time?

Progress bar? Nah does not solve the multiple click problem

How about if we go and disable all the elements while we run the calculations?
PITB. Have to get a reference to each and set enabled to false.

How about if we pop a Dialog that indicates that we are busy so that the user cant touch other buttons? When thread is finished it stops that UI?

Add private variable

```
private ProgressDialog myProgressDialog;
```

and a couple of methods to start and stop it

```
private void progressDialog_start() {
        myProgressDialog = new ProgressDialog(this);
        myProgressDialog.setTitle("Please wait");
        myProgressDialog.setMessage("Notice user cannot interact with rest of UI\
nincluding starting additional threads");
        myProgressDialog.setCancelable(false);
        myProgressDialog.show();
    }

    private void progressDialog_stop(){
        myProgressDialog.dismiss();
    }
```

start and stop in constructor and in runonuithread in thread

---

stop here

so what if the thread goes on forever and we want to cancel it?
Make the thread a member variable of activity
Add a cancel button to progress dialog
Add an onclick handler, if user clicks call `mt.cancel(true)`

```
private void progressDialog_start() {
  myProgressDialog = new ProgressDialog(this);
  myProgressDialog.setButton(DialogInterface.BUTTON_NEGATIVE,
"Cancel", new DialogInterface.OnClickListener() {
      @Override
      public void onClick(DialogInterface dialog, int which) {
        mt.cancel(true);
        dialog.dismiss();
      }
    }
  );
  myProgressDialog.setTitle("Please wait");
  myProgressDialog.setMessage("Notice user cannot interact with
rest of UI\nincluding starting additional threads");
  myProgressDialog.setCancelable(false);
  myProgressDialog.show();
}
```

```
tidy up run on UpdateTask thread so it checks for canceled and reacts accordingly
@Override
public void run() {
   super.run();
   for (int i = 0; i <= numberInstances; i++) {
      act.runCalcs(1);
      if (this.iscanceled)
         break;
   }

   //act=null;    //simulates Activity being detached after above line but before next
   act.runOnUiThread(new Runnable() {
      @Override
      public void run() {
         act.progressDialog_stop();
         if(UpdateTask.this.iscanceled)
            act.binding.mainContainer.textView.setText(USER_CANCELED);
         else
             act.binding.mainContainer.textView.setText(DONE);
      }
   });
}
```

```
================================================================
But… Our thread is holding onto activity via member var, what happens if phone
rotates? (start then rotate look at logcat, leaked window…)
E/WindowManager: android.view.WindowLeaked: Activity
com.example.demoasynctask.MainActivity has leaked window DecorView@a8aeb3a[Please
wait] that was originally added here
        at android.view.ViewRootImpl.<init>(ViewRootImpl.java:597)
```

Plus you crash since returned thread wants to update ui that aint there. Plus, not checking for act==null and even if you do there is a race condition between the null check and the dereference

```
if(act)
        act.runCalcs(…)
```

Attempted Solution- Host AsyncTask in ViewModel

update build.gradle

```
implementation "androidx.lifecycle:lifecycle-viewmodel"
```

Fix… Move thread to App with a Viewmodel, let ViewModel hosts thread

add a ViewModel class

```
package com.example.asynctaskpdialog_inclass; //whatever your package name is here
```

```
import androidx.lifecycle.ViewModel;

public class DataVM extends ViewModel {

}
```

Move AsyncTask into ViewModel
now some stuff changes in main, progress dialog functions for instance need public access

add a ViewModel member to MainActivity

```
//persists accross config changes
DataVM myVM;
```

and in onCreate add the following

```
// 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(this).get(DataVM.class);
```

launch thread from ViewModel

```
fab.setOnClickListener(new View.OnClickListener() {
      //a better (but still bad) way
      @Override
      public void onClick(View v) {
        //what is this funkyness? create a new instance of the thread and attach to
        //myVM.mt
        myVM.mt= myVM.new AST_task(MainActivity.this);
        myVM.mt.execute(NUMBER_UPDATES);
      }

    });
```

But… still tied directly to activity.. rotate the phone and you keep the thread but what happens to act?

Simple but misguided ?
Have attach and detach in thread
Mainactivity attaches in onCreate and detaches in OnDestroy
but now I have to do a LOT of null checks AND I am still mixing my UI with my threads, a recipe for confusion
AND I am not threadsafe… Where is the problem in the following?

```
protected void onPostExecute(Void aVoid) {
  super.onPostExecute(aVoid);
  if (act != null) {
    act.tv.setText(DONE + Integer.toString(numberInstances--));
    act.progressDialog_stop();
  }
}
```

what if activity destroyed after null check? Or progressdialog for that matter?

How about if we just seperate them totally…

Enter LiveData – An observer pattern where all the data is kept in the ViewModel and the Activity is notified when the data changes, the framework correctly manages all the lifecycle events for you.

In ViewModel (DataVM)

Get rid of the dependency on the Activity in the ViewModel and/or the AsyncTask

Add MutableLiveData listeners and initialize in DataVM's constructor

```
public DataVM() {
   super();
   //initialize as appropriate
   cntr = new MutableLiveData<Integer>();
   cntr.setValue(0);    //initialize
}

//lets add some livedata
private MutableLiveData<Integer> cntr ;
public MutableLiveData<Integer> getCurrentProgress() {
   return cntr;
}
```

In ViewModels Thread (DataVM.MyThread)

```
cntr.postValue(i);  //update the livedata from a seperate thread
```

In MainActivity, Add an observer in OnCreate

```
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   tv = findViewById(R.id.textView2);
   bStart= findViewById(R.id.bStart);
   bCancel = findViewById(R.id.bCancel);
   pBar = findViewById(R.id.progressBar1);
   // 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(this).get(DataVM.class);
   // Create the observer which updates the UI.
   final Observer<Integer> cntrObserver = new Observer<Integer>() {
      @Override
      public void onChanged(@Nullable final Integer newInt) {
         // Update the UI, in this case, a TextView.
         tv.setText("The new cnt=" +Integer.toString(newInt));
         pBar.setProgress(newInt);
      }
```

```
};
//now observe
myVM.getCurrentProgress().observe(this,cntrObserver);
```

You are done. See ViewModel_LiveData_Thread project.