What we Have
Everything in MainActivity

MainActivity

```
//static inner class
Public static class AddTask extends AsyncTask…
:
//class member
AddTask MyTask;
:
//start the asynctask
myTask.Execute()
```

But:

1. **AddTask has an explicit reference to MainActivity**
2. **If you forget the static, then you have an implicit reference**
3. **So you have to manage attaching and detaching to the host activity, and you are relying on OnRetainNonConfigurationInstance()**

Doesn't scale so….

Possible Solution
Move AsyncTask to ViewModel

MainActivity

```
//class member
DataVM myVM;
:
myVM.mt=myVM.new AddTask(MainActivity.this)
```

DataVM

```
// inner class
Public class AddTask extends AsyncTask…
:
//class member
AddTask mt;
:
//start the asynctask
mt.Execute()
```

Good:

1. **Does not rely on OnRetainNonConfigurationInstance()**

Bad

1. **What happens when phone rotates? Mainactivity is torn down and recreated.**
2. **How do you verify the activity mt uses is valid? How do you avoid race conditions?**

Possible Solution
Move AsyncTask to ViewModel

MainActivity

```
//class member
DataVM myVM;
:
myVM.mt=myVM.new AddTask(MainActivity.this)
```

DataVM

```
// inner class
Public class AddTask extends AsyncTask…
:
//class member
AddTask mt;
:
//start the asynctask
mt.Execute()
```

Good:

1. **Does not rely on OnRetainNonConfigurationInstance()**

Bad

1. **What happens when phone rotates? Mainactivity is torn down and recreated.**
2. **How do you verify the activity mt uses is valid? How do you avoid race conditions?**
3. **Also DataVM.mt is heavily entwined (coupled) with MainActivity**

Another Possible Solution
Use ViewModel and LiveData

MainActivity

```
// 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,
        pBar.setProgress(newInt);
    }
};
//now observe
myVM.getCurrentProgress().observe( owner: this,cntrObserver);
```

Mainactivity askes to be
Notifies when cntr changes

DataVM

```
private MutableLiveData<Integer> cntr;
public private MutableLiveData<Integer>
        getCurrProgress(){return cntr;}


// inner class
Public class AddTask extends AsyncTask…
{
  :
  … doInBackground(..){
     cntr.postValue(3);
     }
}
:
//class member
AddTask mt;
:
//start the asynctask
mt.Execute()
```

Another Possible Solution
Use ViewModel and LiveData

MainActivity

```
// 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,
        pBar.setProgress(newInt);
    }
};
//now observe
myVM.getCurrentProgress().observe( owner: this,cntrObserver);
```

DataVM

```
private MutableLiveData<Integer> cntr;
public private MutableLiveData<Integer>
        getCurrProgress(){return cntr;}


// inner class
Public class AddTask extends AsyncTask…
{
  :
  … doInBackground(..){
    cntr.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the asynctask
mt.Execute()
```

This line updates cntr from the thread

Another Possible Solution
Use ViewModel and LiveData

MainActivity

```
// 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,
        pBar.setProgress(newInt);
    }
};
//now observe
myVM.getCurrentProgress().observe( owner: this,cntrObserver);
```

Which results in this onChanged method being called, ViewModel and LiveData Autohandle all MainActivity changes

DataVM

```
private MutableLiveData<Integer> cntr;
public private MutableLiveData<Integer>
        getCurrProgress(){return cntr;}


// inner class
Public class AddTask extends AsyncTask…
{
  :
  … doInBackground(..){
     cntr.postValue(3);
     }
}
:
//class member
AddTask mt;
:
//start the asynctask
mt.Execute()
```

Another Possible Solution
Use ViewModel and LiveData

MainActivity

```
// 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,
        pBar.setProgress(newInt);
    }
};
//now observe
myVM.getCurrentProgress().observe( owner: this,cntrObserver);
```

PRESTO!
Complete decoupling
MainActivity is updated
whenever a change occurs

No coupling between
ViewModel and Activity
Everybody wins

DataVM

```
private MutableLiveData<Integer> cntr;
public private MutableLiveData<Integer>
        getCurrProgress(){return cntr;}


// inner class
Public class AddTask extends AsyncTask…
{
  :
  … doInBackground(..){
    cntr.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the asynctask
mt.Execute()
```