

What we Have

Everything in MainActivity

MainActivity

```
//static inner class
Public static class AddTask extends AsyncTask...
:
//class member
AddTask MyTask;
:
//start the async task
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

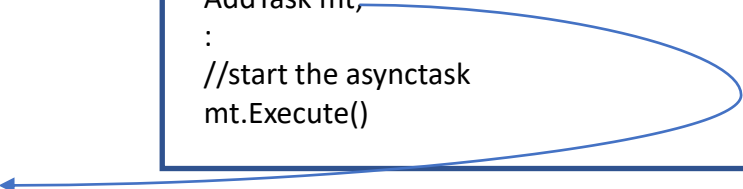
Move AsyncTask to ViewModel

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 async task  
mt.Execute()
```



Good:

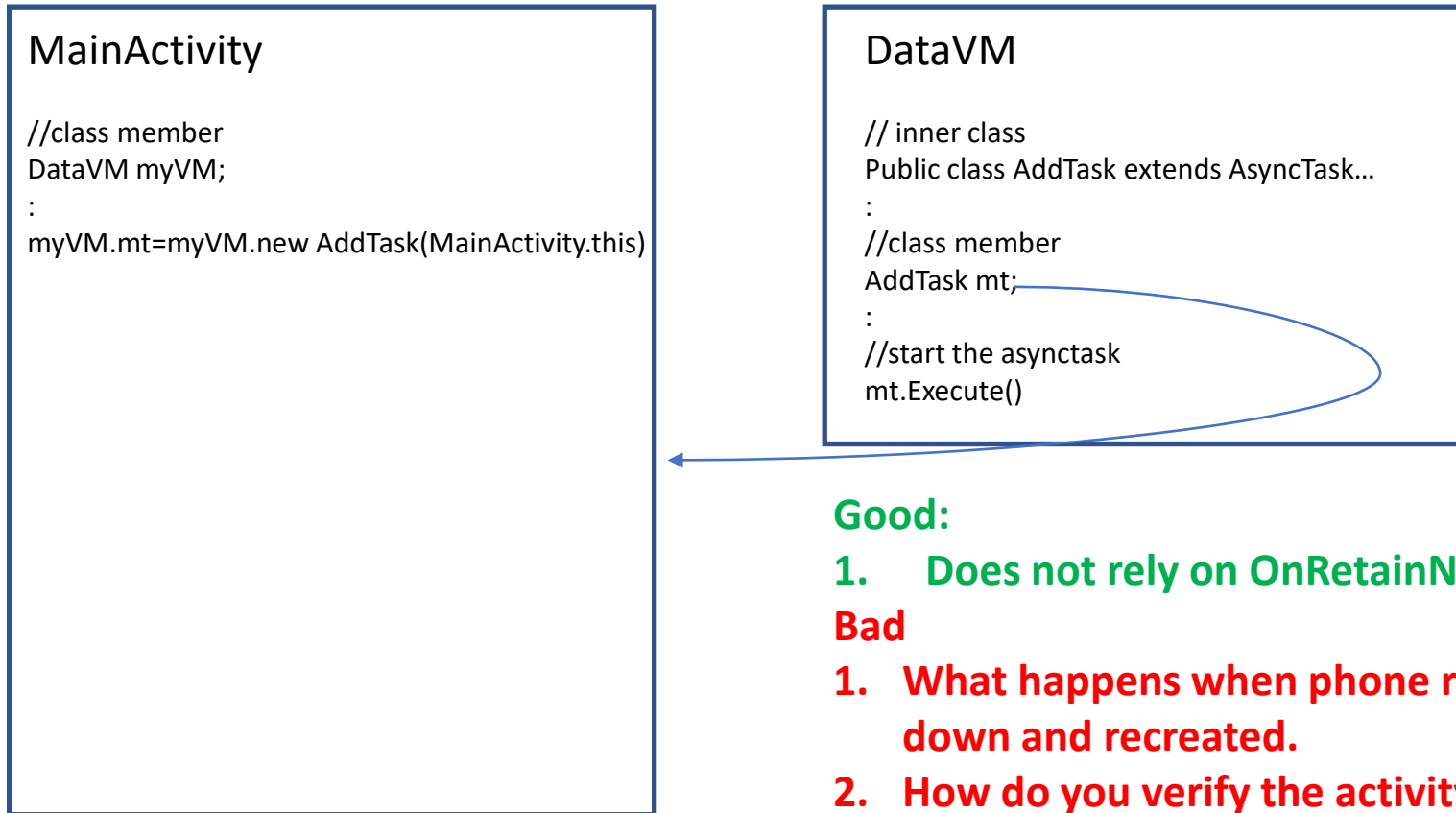
1. Does not rely on `OnRetainNonConfigurationInstance()`

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



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> cntrobsvr = new Observer<Integer>() {
    @Override
    public void onChanged(@Nullable final Integer newInt) {
        // Update the UI,
        progressBar.setProgress(newInt);
    }
};
//now observe
myVM.getCurrentProgress().observe( owner: this, cntrobsvr);
```

Mainactivity asks to be
Notifies when cnt changes



DataVM

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

// inner class
Public class AddTask extends AsyncTask...
{
    :
    ... doInBackground(..){
        cnt.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the async task
mt.execute()
```

Another Possible Solution

Use ViewModel and LiveData

MainActivity

```
// Create the observer which updates the UI.
final Observer<Integer> cntrobsvr = new Observer<Integer>() {
    @Override
    public void onChanged(@Nullable final Integer newInt) {
        // Update the UI,
        progressBar.setProgress(newInt);
    }
};
//now observe
myVM.getCurrentProgress().observe( owner: this, cntrobsvr);
```

DataVM

```
private MutableLiveData<Integer> cnt;
public private MutableLiveData<Integer>
    getCurProgress(){return cnt;}

// inner class
Public class AddTask extends AsyncTask...
{
    :
    ... doInBackground(..){
        cnt.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the async task
mt.Execute()
```

← This line updates cnt from the thread

Another Possible Solution

Use ViewModel and LiveData

MainActivity

```
// Create the observer which updates the UI.
final Observer<Integer> cntrobsvr = new Observer<Integer>() {
    @Override
    public void onChanged(@Nullable final Integer newInt) {
        // Update the UI,
        progressBar.setProgress(newInt);
    }
};
//now observe
myVM.getCurrentProgress().observe(owner: this, cntrobsvr);
```

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

DataVM

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

```
// inner class
Public class AddTask extends AsyncTask...
{
    :
    ... doInBackground(..){
        cnter.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the async task
mt.Execute()
```

Another Possible Solution

Use ViewModel and LiveData

MainActivity

```
// Create the observer which updates the UI.
final Observer<Integer> cntrobsvr = new Observer<Integer>() {
    @Override
    public void onChanged(@Nullable final Integer newInt) {
        // Update the UI,
        progressBar.setProgress(newInt);
    }
};
//now observe
myVM.getCurrentProgress().observe( owner: this, cntrobsvr);
```

PRESTO!
Complete decoupling
MainActivity is updated
whenever a change occurs

No coupling between
ViewModel and Activity
Everybody wins

DataVM

```
private MutableLiveData<Integer> cnt;
public private MutableLiveData<Integer>
    getCurProgress(){return cnt;}
```

```
// inner class
Public class AddTask extends AsyncTask...
{
    :
    ... doInBackground(..){
        cnt.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the async task
mt.Execute()
```