Everything in MainActivity

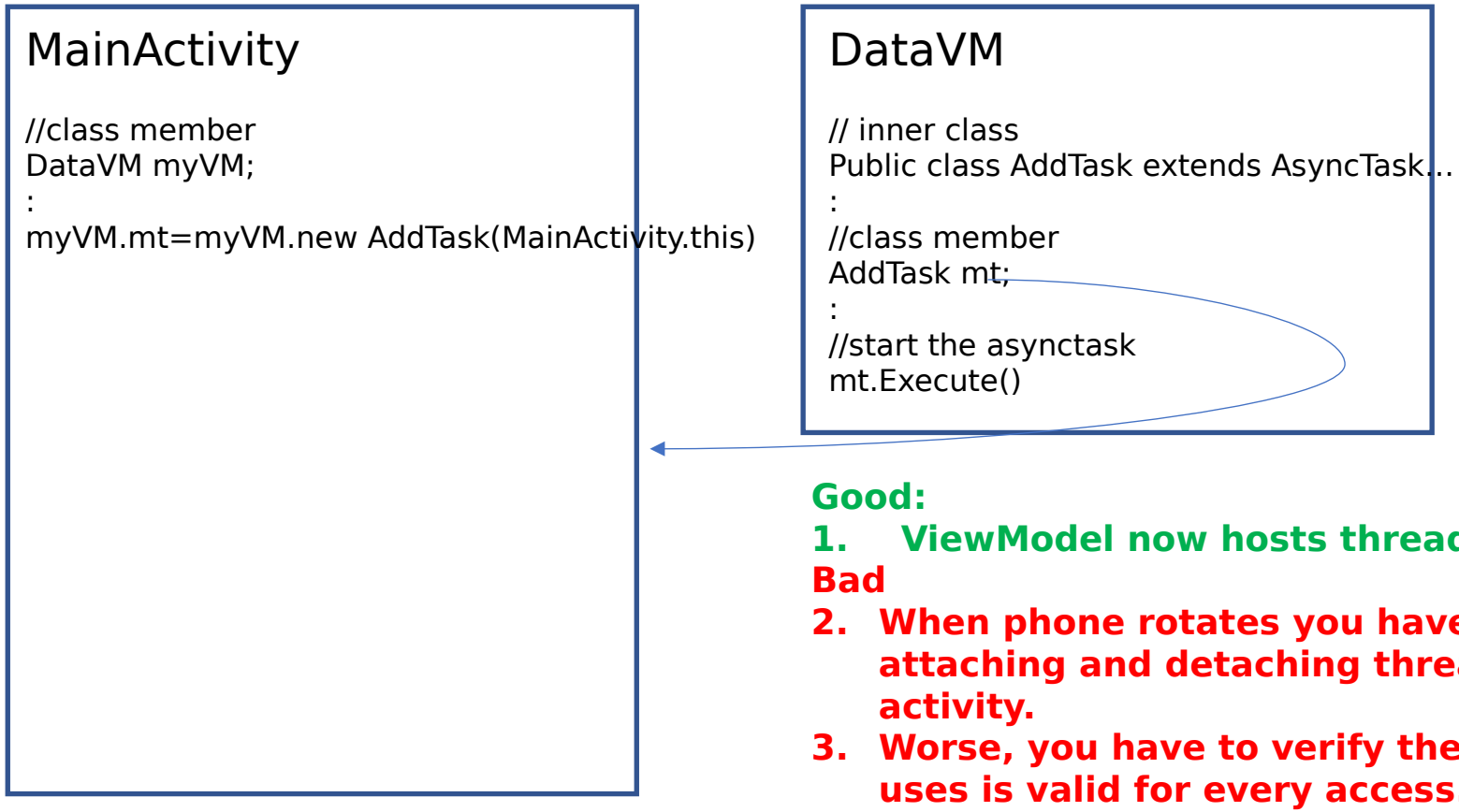
MainActivity

```
//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 to enclosing activity so it cannot be GC’ed until thread exits
3. For Rotations, how do you pass thread to new activity?

Possible Solution
Move Thread to ViewModel



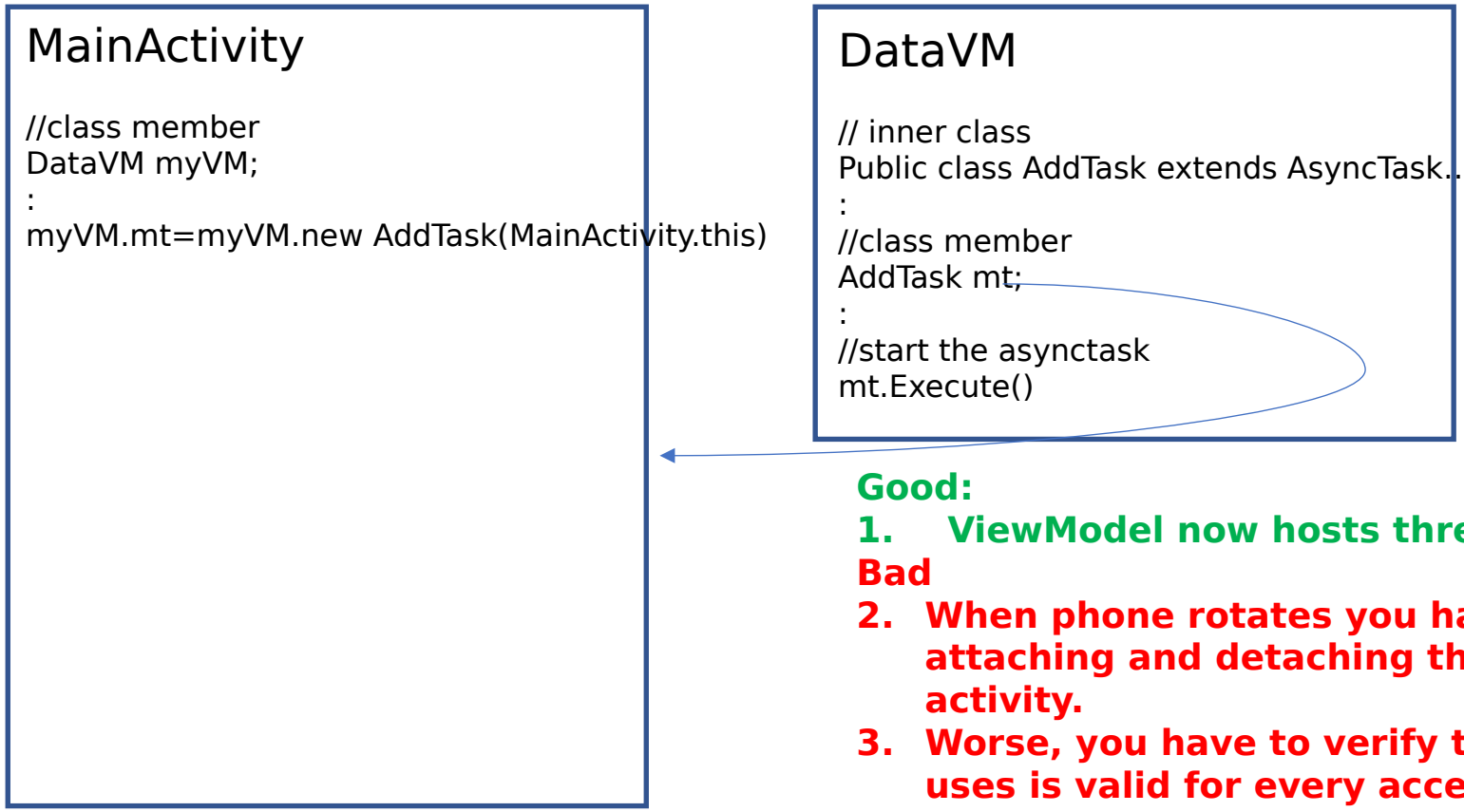
Good:

1. **ViewModel now hosts thread**

Bad

2. **When phone rotates you have to handle attaching and detaching thread to activity.**
3. **Worse, you have to verify the activity your thread uses is valid for every access. How do you avoid race conditions?**

Possible Solution
Move Thread to ViewModel



Good:

1. **ViewModel now hosts thread**

Bad

2. **When phone rotates you have to handle attaching and detaching thread to activity.**
3. **Worse, you have to verify the activity your thread uses is valid for every access. How do you avoid race conditions?**
4. **Also DataVM.mt is heavily coupled with MainActivity**

Better 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 asks to be
Notified 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()
```

Better 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

Better 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()
```

Better 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()
```

Better 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

But AsyncTask was designed
To have high coupling
with activity

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()
```

Best 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

So use a Java Thread Instead

DataVM

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


// inner class
public class AddTask extends Thread {
:
}

:
//class member
AddTask mt;
:
//start the asynctask
mt.Execute()
```