

# Everything in MainActivity

## MainActivity

```
//static inner class
Public static class AddTask extends Thread...
:
//class member
AddTask MyTask;
:
//start the thread
myTask.start()
```

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

### MainActivity

```
//class member
DataVM myVM;
:
//create a new thread
//myVM will host it
myVM.mt=myVM.new AddTask(MainActivity.this)
```

### DataVM

```
// inner class
Public class AddTask extends Thread...
:
//class member
AddTask mt;
:
//start the thread
mt.start()
```

#### 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.**
4. **Also, how do you avoid race conditions?**

# Possible Solution

## Move Thread to ViewModel

### MainActivity

```
//class member
DataVM myVM;
:
//create a new thread
//myVM will host it
myVM.mt=myVM.new AddTask(MainActivity.this)
```

### DataVM

```
// inner class
Public class AddTask extends Thread...
:
//class member
AddTask mt;
:
//start the thread
mt.start()
```

#### 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.**
4. **Also, how do you avoid race conditions?**
5. **Also DataVM.mt is heavily coupled with MainActivity**

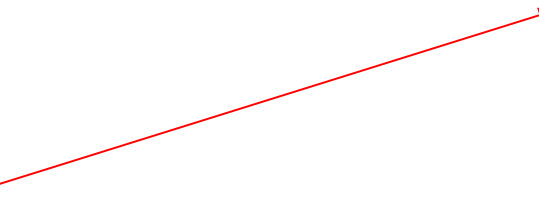
# Better Solution

## Use ViewModel and LiveData

### MainActivity

```
// Create the observer which updates the UI.
final Observer<Integer> 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  
Notified when cnt changes



### DataVM

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

```
// inner class
Public class AddTask extends Thread...
{
    :
    ... run(){
        cnt.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the thread
mt.start();
```

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

// inner class

Public class AddTask extends Thread...

{

:

```
... run(){
    cnter.postValue(3);
}
```

This line updates cnter  
from the thread,  
if updating from UI  
Thread, use setValue

}

:

//class member

AddTask mt;

:

//start the thread

mt.start();

# Better 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,

### DataVM

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

```
// inner class
Public class AddTask extends Thread...
{
    :
    ... run(){
        cnter.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the thread
mt.start();
```

# Better 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  
In LiveData

No coupling between  
ViewModel and Activity  
Everybody wins

### DataVM

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

```
// inner class
Public class AddTask extends Thread...
{
    :
    ... run(){
        cnter.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the thread
mt.start();
```

# Better 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  
In LiveData

No coupling between  
ViewModel and Activity  
Everybody wins

This is one reason why  
AsyncTask was dropped.  
It was designed  
To have high coupling  
with the activity

### DataVM

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

```
// inner class
Public class AddTask extends Thread...
{
    :
    ... run(){
        cnter.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the thread
mt.start();
```



# Better 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  
In LiveData

No coupling between  
ViewModel and Activity  
Everybody wins

This is one reason why  
AsyncTask was dropped.  
It was designed  
To have high coupling  
with the activity

**Thats why Java Threads  
Are used instead**

### DataVM

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

```
// inner class
Public class AddTask extends Thread...
{
    :
    ... run(){
        cnter.postValue(3);
    }
}
:
//class member
AddTask mt;
:
//start the thread
mt.start();
```