# CS 371 Laboratory 9: Threads

This lab is designed to help you understand threads and how to use them in Java.

## Preliminaries

### Checkout the Starter Code

Clone the GitHub repository: https://github.com/cs301up/CS371ThreadsLabStarter and load the project into Android Studio.

### A Note on Android Java and Threads

Bad things can happen when two different threads try to modify the same GUI object at the same time. To prevent this, Android OS has a rule that only the thread that created a GUI object can modify it. But is it common for a program in a different thread to want to, for example, draw on a SurfaceView. In order to allow this to happen, it needs to do a “post”—which sends a message to the original thread to perform the operation. This operation is scheduled but does not actually run until the thread becomes available.

The standard way to visually update a SurfaceView is to:

* Modify whatever data objects the SurfaceView will be examining in order to do its drawing.
* Perform a “postInvalidate” on the SurfaceView. This tells the SurfaceView that its view is out of date, so that it should redraw itself. It does it by doing a “post”, so that the ‘draw’ method is run in the SurfaceView’s thread.
* The SurfaceView will redraw itself once the thread becomes available.

### Run the Program

Verify that the app runs correctly in Android Studio. You should see a black box full of “stars” as well as a button and a seekbar at the bottom of the screen. Each time you press the button you should see a new frame in an animation of twinkling stars. The seekbar changes the brightness of the stars.

Note: The function of the button and seekbar will change as the lab progresses.

### Examine the Java Source Code

You’ll be working directly with this source code in the lab so take time now to examine each of the source code files and make sure you understand each method that you find therein.

* MainActivity.java initializes the animation view and responds to user input
* AnimationView.java stores a list of animations that it displays
* Animation.java is an abstract class. Each animation subclasses this.
* StarAnimation.java defines the twinkling stars animation you saw when you ran the app.
* Star.java defines a single star from the twinkling stars animation.

## Laboratory

### Part 1: Make the Animation Run Automatically

If you examine the onClick() method in MainActivity, you can see that each time the user presses the button, the program calls postInvalidate() upon the AnimationView. This causes the AnimationView to redraw itself and, thus, display the next frame in the animation.

One way to automate the animation is to modify the code so that it will repeatedly call postInvalidate(). Modify the code in this manner - **without using threads for now** - and then test it to verify that it works. Your program must animate without you clicking the Button even once.

**checkpoint 1 (15 points): Show your code to the instructor.**

### Part 2: Slow down the animation with Thread.sleep()

Using your favorite web browser, examine the documentation for the Thread class in java (java.lang.Thread). In particular, examine the sleep() method and its function. Use this method to modify your program so that the animation runs very slowly: one frame per 3 seconds.

What effect does this addition have on the behavior of your program particularly in terms of the button and seekbar?

**checkpoint 2 (15 points): Show your instructor your code and the “sleepy” animation and demonstrate the problem it causes with the button and seekbar.**

### Part 3: Create a thread for animation

To fix the issue, we’ll need to drive the animation with a separate thread. Follow these steps:

1. Remove the code you added in the previous checkpoints
2. Create a new class called ThreadedAnimationView in the project that is a subclass of Thread.
3. The class should have an instance variable to hold a reference to the app’s AnimationView object. Initialize the instance variable in the constructor with the AnimationView argument passed in.
4. Override the run() method from the parent class. This method should contain an infinite loop that performs the following actions:
   1. calls postInvalidate() on the AnimationView
   2. sleeps for *three* seconds
5. In the onCreate() method in the MainActivity, create an instance of this class and then call the start() method on this instance.

Test your program and verify it behaves properly.

**checkpoint 3 (15 points): Show your instructor your code and the working animation, and demonstrate that the problem with the button and seekbar has been resolved. Explain to your instructor why this is so.**

### Part 4: Adding and Removing Stars

Modify your animation so that it sleeps for 1/20th of a second between “redraws” rather than for three seconds.

Currently, the seekbar changes the brightness of the stars. Modify the code in the StarAnimation class so that the seekbar can be used to increase or decrease the total number of stars on the field instead of controlling the brightness. (Note that addStar() and removeStar() methods have already been written for you.) Write your code so that there will always be at least 100 stars but never more than 1000 stars. Test your program and verify that it works as you intend.

**checkpoint 4 (10 points): Show your instructor your code and the app’s new functionality.**

### Part 5: Using a Thread to Add and Remove Stars

Now create a new thread class that automatically adds or removes a star (randomly chosen) from a given StarAnimation object every 40 milliseconds. Start an instance of this thread in the StarAnimation constructor. Verify by observation that stars are being added and removed.

Allow your program to run for a while. It should eventually crash with a ConcurrentModificationException. You may be able to speed up this occurrence by rapidly changing the number of stars on the field using the seekbar. If you still can’t get it to crash, try reducing the length of time in your calls to Thread.sleep.

Sometimes, you have sections of code that you never want to have execute simultaneously in different threads because they might interfere with each other. The term for this is a *critical section*. Your program is crashing because of its critical sections. Analyze your code and decide where the critical sections are.

**checkpoint 5 (25 points): Show your instructor or lab assistant your code and the resulting stack trace. Then, explain what you think is happening and identify the critical section(s) in your code.**

### Part 6: Protecting your critical sections with synchronized

You can “protect” a critical section using the synchronized reserved word in Java. When you use this reserved word, you must specify an object used by the section. Here is what it looks like:

synchronized(myObject) {

<critical section code goes here>

}//end of critical section

Any section of code surrounded in this way will not be executed simultaneously by two or more threads (as long as they are synchronizing on the same object). If a thread reaches a synchronized section while another thread is executing it, then it will wait until the other thread has completed that section.

Also, if two *different* critical sections are synchronized using the same object, then they also are mutually exclusive with each other.

Using the synchronized reserved word, protect the critical sections in your app. Since there is overhead associated with the use of synchronized, you don’t want to use it willy-nilly and synchronize sections everywhere or synchronize large blocks of code that don’t need to be synchronized. Instead, use synchronize efficiently and effectively and put it only where necessary.

In order to intensively test your changes, make sure that it works when the time on the “add and remove stars” thread is changed from 40 milliseconds to 2 milliseconds.

**checkpoint 6 (20 points): Show your instructor or lab assistant your code and demo that your program no longer crashes.**