Thread Methods

CSCI 201
Principles of Software Development

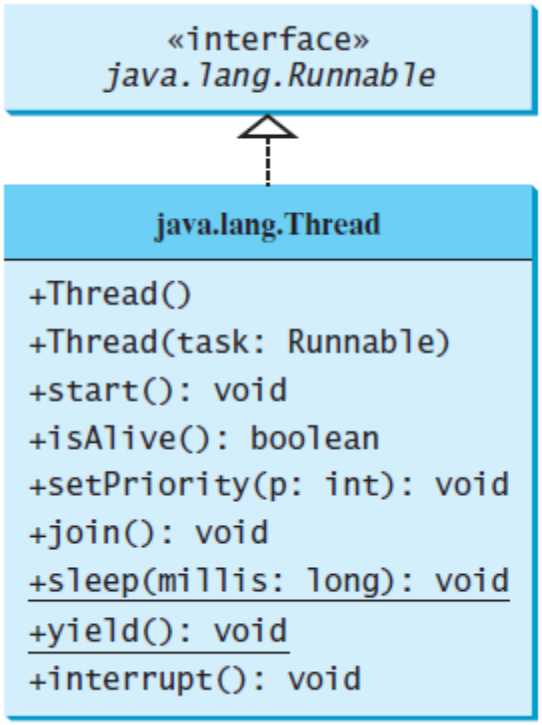
Jeffrey Miller, Ph.D.
jeffrey.miller@usc.edu

USC Viterbi
School of Engineering

Outline

- Thread Methods
- Program

Thread Class



Creates an empty thread.	
Creates a thread for a specified task.	
Starts the thread that causes the `run()` method to be invoked by the JVM.	
Tests whether the thread is currently running.	
Sets priority `p` (ranging from 1 to 10) for this thread.	
Waits for this thread to finish.	
Puts a thread to sleep for a specified time in milliseconds.	
Causes a thread to pause temporarily and allow other threads to execute.	
Interrupts this thread.	

Creating and Starting Threads

- To create a thread, you need to instantiate the `Thread` class
  - The `Thread` class has a default constructor
  - The `Thread` class has a constructor that takes a `Runnable` object
- The class must have overridden the `run()` method from the `Runnable` interface
- To start the thread, call `start()`
  - The `run()` method will get called once you start the thread
  - In other words, the `run()` method will be running *simultaneously* with the other threads

`yield()` and `sleep()` Methods

- The `yield()` method temporarily releases the thread's current time in the CPU to another thread
  - The actual behavior of `yield()` is that the thread is moved from the running state back to the ready state
    - The OS then decides which thread to put back into the CPU, which could be the same thread again if there are no other threads waiting in the ready state
- The `sleep(long milliseconds)` method puts the thread to sleep for a certain number of milliseconds
  - The thread will be moved into the ready state after the number of milliseconds has elapsed, so the thread will sleep for *at least* the number of milliseconds
  - The `sleep(long milliseconds)` can throw an `InterruptedException`

Sleeping Thread Example

- If an `InterruptedException` is thrown, the thread **should terminate** because that signals that another thread is requesting for it to stop what it is currently executing

```
1   public class Test {
2     public static void main(String[] args ) {
3       System.out.println("First line");
4       TestThread ta = new TestThread('a');
5       TestThread tb = new TestThread('b');
6       TestThread tc = new TestThread('c');
7       ta.start();
8       tb.start();
9       tc.start();
10      System.out.println("Last line");
11    }
12  }
13  class TestThread extends Thread {
14    private char c;
15    public TestThread(char c) {
16      this.c = c;
17    }
18    public void run() {
19      for (int i=0; i < 20; i++) {
20        System.out.print(i + "" + c + " ");
21        try {
22          Thread.sleep(1000);
23        } catch (InterruptedException ie) {
24          System.out.println("interrupted");
25        }
26      }
27      System.out.println("");
28    }
29  }
```

```
// preferred way of doing interrupts since you
// want the thread to terminate on an interrupt
18    public void run() {
19      try {
20        for (int i=0; i < 20; i++) {
21          System.out.print(i + "" + c + " ");
22          Thread.sleep(1000);
23        }
24        System.out.println("");
25      } catch (InterruptedException ie) {
26          System.out.println("interrupted");
27      }
28    }
```

join() method

- The `join()` method allows one thread to specify that it wants to wait for another thread to complete before it continues to execute

```
1   public class Test {
2     public static void main(String[] args ) {
3       System.out.println("First line");
4       TestThread ta = new TestThread('a');
5       TestThread tb = new TestThread('b');
6       TestThread tc = new TestThread('c');
7       ta.start();
8       tb.start();
9       tc.start();
10      try {
11        tc.join();
12      } catch (InterruptedException ie) {
13        System.out.println("InterruptedException: " + ie.getMessage());
14      }
15      System.out.println("Last line");
16    }
17  }
18
19  class TestThread extends Thread {
20    private char c;
21    public TestThread(char c) {
22      this.c = c;
23    }
24    public void run() {
25      for (int i=0; i < 20; i++) {
26        System.out.print(i + "" + c + " ");
27      }
28      System.out.println("");
29    }
30  }
```

```
First line
0a 1a 2a 3a 4a 5a 6a 7a 8a 9a 10a 11a 12a 13a 14a 15a 16a 17a 18a 19a
0b 1b 2b 3b 4b 5b 6b 7b 8b 9b 10b 11b 12b 13b 14b 15b 16b 17b 18b 19b
0c 1c 2c 3c 4c 5c 6c 7c 8c 9c 10c 11c 12c 13c 14c 15c 16c 17c 18c 19c
Last line
```

```
First line
0c 1c 2c 3c 0a 4c 1a 5c 2a 6c 3a 7c 4a 8c 5a 6a 7a 8a 9a 10a 11a 9c 12a 10c 13a 11c 14a 12c 13c 14c 15c 16c 17c 18c 19c
15a 16a 17a 18a 19a
Last line
0b 1b 2b 3b 4b 5b 6b 7b 8b 9b 10b 11b 12b 13b 14b 15b 16b 17b 18b 19b
```

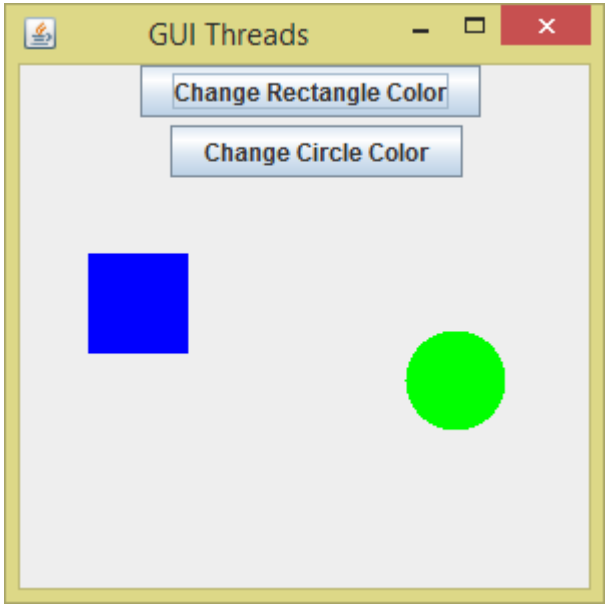
```
First line
0a 0b 1b 2b 3b 4b 5b 6b 7b 8b 9b 10b 11b 12b 13b 14b 15b 16b 17b 18b 19b
0c 1c 2c 3c 4c 1a 2a 3a 4a 5a 6a 7a 8a 9a 10a 11a 12a 13a 14a 15a 16a 17a 18a 19a
5c 6c 7c 8c 9c 10c 11c 12c 13c 14c 15c 16c 17c 18c 19c
Last line
```

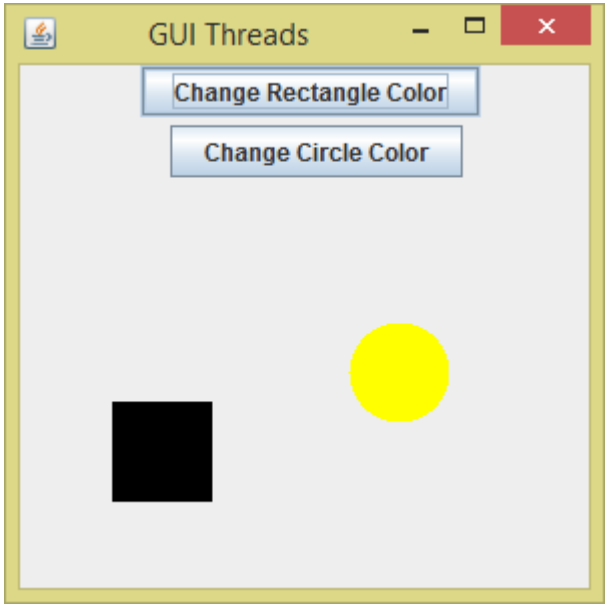
Outline

- Thread Methods
- Program

Program

- Write a program that displays a GUI with two buttons on it. There are also two shapes that will move in different paths. When the buttons are clicked, the color of the shape should change. Each shape will need to be in its own thread so its path can be different, and the GUI will be in its own thread.





Program

- Write a program that displays a GUI with a label on it. Have a button that toggles whether the input affects the label. If the button is set to allow input, display the text the user has typed on the command line. If the button is set to not allow input, do not display the text the user has typed.