Outline

- Thread Methods
- Program
Thread Class

+Thread()
+Thread(task: Runnable)
+start(): void
+isAlive(): boolean
+setPriority(p: int): void
+join(): void
+sleep(millis: long): void
+yield(): void
+interrupt(): void

- 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 in 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
    - This is part of the unpredictable behavior of multi-threaded programming

- 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`
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

```java
public class Test {
    public static void main(String[] args) {
        System.out.println("First line");
        TestThread ta = new TestThread('a');
        TestThread tb = new TestThread('b');
        TestThread tc = new TestThread('c');
        ta.start();
        tb.start();
        tc.start();
        System.out.println("Last line");
    }
}

class TestThread extends Thread {
    private char c;
    public TestThread(char c) {
        this.c = c;
    }
    public void run() {
        try {
            for (int i=0; i < 20; i++) {
                System.out.print(i + " + " + c + " ");
                sleep(1000);
            }
            System.out.println("interrupted");
        } catch (InterruptedException ie) {
            System.out.println("interrupted");
        }
        System.out.println("");
    }
}
```

// preferred way of doing interrupts since you want the thread to terminate on an interrupt
```java
    public void run() {
        try {
            for (int i=0; i < 20; i++) {
                System.out.print(i + " + " + c + " ");
                sleep(1000);
            }
            System.out.println("interrupted");
        } catch (InterruptedException ie) {
            System.out.println("interrupted");
        }
        System.out.println("");
    }
```
### 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.

```java
public class Test {
    public static void main(String[] args) {
        System.out.println("First line");
        TestThread ta = new TestThread('a');
        TestThread tb = new TestThread('b');
        TestThread tc = new TestThread('c');
        ta.start();
        tb.start();
        tc.start();
        try {
            tc.join();
        } catch (InterruptedException ie) {
            System.out.println("InterruptedException: " + ie.getMessage());
        }
        System.out.println("Last line");
    }
}

class TestThread extends Thread {
    private char c;
    public TestThread(char c) {
        this.c = c;
    }
    public void run() {
        for (int i=0; i < 20; i++) {
            System.out.print(i + " " + c + " ");
        }
        System.out.println("");
    }
}
```

Thread Methods
Outline

- Thread Methods
- 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 JLabel on it. Have a button that toggles whether the input affects the JLabel. 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.