# 作业 1 报告

韩云飞 SA13226297

Please write two java multi-threaded programs according to what you have learned in the class. Then use the output of your programs to explain the following issues :

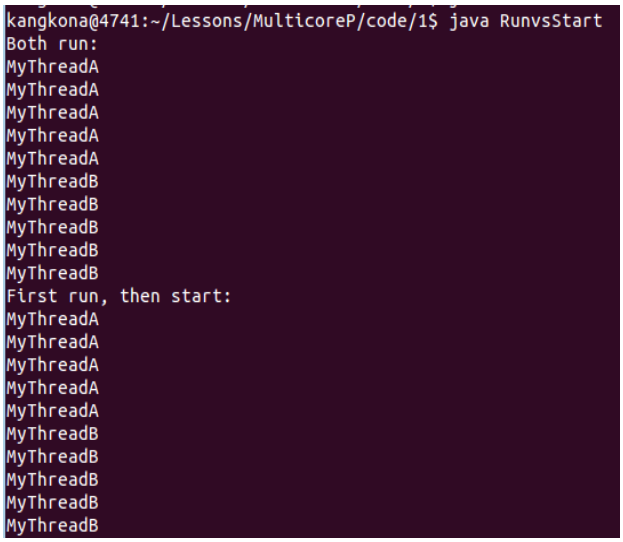
1) The difference between the methods “run()” and “start()”.
2) The function of “join()” method, how to use “join()” to avoid nondetermistic output?
3) You need to figure out the threads that may be executed concurrently in your programs.
4) You need to figure out the shared and local memory in your programs and how do they affect the results of your programs.

RunvsStart.java 程序用来演示 run 与 start 区别：

```
//RunvsStart.java
import java.util.Random;
class MyThreadA extends Thread {
	public  void run() { // entry point for thread
		try {
			Random rd = new Random(7);
			for(int i = 5; i > 0; i--) {
			System.out.println("MyThreadA");
			Thread.sleep(rd.nextInt(100));
			}
		}
		catch(InterruptedException e) {
			System.out.println("Interrupted.");
		}
	}
}

class MyThreadB extends Thread {
	public  void run() { // entry point for thread
		try {
			Random rd = new Random(10);
			for(int i = 5; i > 0; i--) {
			System.out.println("MyThreadB");
			Thread.sleep(rd.nextInt(100));
			}
		}
	 catch(InterruptedException e) {
		System.out.println("Interrupted.");
	 }
    }
}

public class RunvsStart {
	public enum ORDER { R2, S2, FRLS, FSLR }
	public  static void order(ORDER order) {
		MyThreadA t1 = new MyThreadA();
		MyThreadB t2 = new MyThreadB();
		switch (order) {
			case R2:
			   t1.run();
			   t2.run();
			   break;
			case S2:
```

```
                t1.start();
                t2.start();
                break;
            case FRLS:
                t1.run();
                t2.start();
                break;
            case FSLR:
                t1.start();
                t2.run();
                break;
            default: break;
        }
    }
    public static void main(String [] args) {
        System.out.println("Both run:");
        order(ORDER.R2);


        System.out.println("First run, then start:");
        order(ORDER.FRLS);

        try {
            Thread.sleep(1000);
        }
        catch(InterruptedException e) {
        System.out.println("Interrupted.");
        }

        System.out.println("First start, then run:");
        order(ORDER.FSLR);

        System.out.println("Both start:");
        order(ORDER.FSLR);
    }
}
```

运行结果为：




分析结果知：
(1) run 每次都在原来的线程中继续执行，start 则是先创建新的线程然后在新线程里面调用 run 执行;
(2) 如果先执行 run，则需要等整个 run 函数运行完才会往下执行;
(3)start 会创建新的线程执行，多个线程会并发执行;
(4)在不影响并行性和正确性的前提下，复用原来的线程可以减少开销(eg：在末尾把 start 换为 run)。

//JoinDemo.java 用来展示 Join 的作用和共享数据与局部数据的交互

```
//JainDemo.java
class SharedData {
	public int a = 0;
	public String s = null;

	public SharedData() {
		a = 10;
		s = "Test";
	}
}

class AddThread  extends Thread {
	private SharedData m_data = null;
	public AddThread(SharedData data) {
		m_data = data;
	}

	public void run() {
		for (int i = 0; i < 5; i++) {
			m_data.a++;
			System.out.println(m_data.a);
		}
	}
}

class SubThread  extends Thread {
	private SharedData m_data = null;
	public SubThread(SharedData data) {
		m_data = data;
	}

	public void run() {
		for (int i = 0; i < 5; i++) {
			m_data.a--;
			System.out.println(m_data.a);
		}
```

```
    }
}

public class JoinDemo {
    public static void main(String[] args) {
        SharedData data = new SharedData();
        AddThread add1 = new AddThread(data);
        SubThread sub1 = new SubThread(data);

        System.out.println("No Join:");
        add1.start();
        sub1.start();
        // for (int i = 0; i < 10; i++) {
        //     System.out.println(data.a);

        // }

        try {
            Thread.sleep(1000);
        }
     catch(InterruptedException e) {
        System.out.println("Interrupted.");
    }


        AddThread add2 = new AddThread(data);
        SubThread sub2 = new SubThread(data);

        System.out.println("With Join:");
        add2.start();
      try {
            add2.join();
        }
       catch(InterruptedException e) {
        System.out.println("Interrupted.");
       }

        sub2.start();
    }
}
```

演示结果为：

分析：
(1)共享数据与局部数据的交互
程序中有两个线程 Add,Sub, Main 的 ShareData 为二者共享，传递给二者的局部数据，
由于二者的局部数据都是对共享数据的引用，所以一个线程修改了共享数据，另一个线程的局部数据也会受到影响。
(2)Join 的作用
在没有 Join 时，Add，Sub 线程是并行执行的，所以输出的结果没有规律的，而且最终的结果无法预测; 在 Add，Sub 之间加入 Add.Join 之后，Sub 线程只能在 Add 执行完之后才会被执行，所以输出结果是先做 5 次加法，再做 5 次减法，最终结果必定为 10。Join 的作用是设置必经节点(Terminator)来控制程序的行为，使得并行程序的结果也能可测。