Assignment 6

Problem 1

Output:-

```
1
1
2
2
3
3
```

The same output is produced if the program is run multiple times because the program is rerun multiple times on the same JVM/platform but may vary if any one these changes. The OS plays a major role in determining the execution order and assignment of thread priorities.

Because of the usage of Thread.sleep(500) , each thread of execution i.e. t1 and t2 sleeps for 0.5 second and allows the other thread to execute/perform its operation.

The order of execution is as follows:-
Instances of TestSleepMethod1 t1 and t2 are created in main.
t1 thread starts.
t2 thread starts.
1)t1(sleeps for 0.5 second) and prints 1.
2)t2 (sleeps for 0.5 second) and prints 1.
3)t1 (sleeps for 0.5 second)and prints 2
4)t2(sleeps for 0.5 second) and prints 2
5)t1(sleeps for 0.5 second)and prints 3
6)t2(sleeps for 0.5 second) and prints 3
7)Loop is completed and the program ends.
try{Thread.sleep(500);}

Problem 2

Incorrect program with the errors(red) and corrections(blue)

```
class TestJoinMethod1 implements extends  Thread{
public void start run(){
for(int i==1;i<=3;i++){
try{
Thread.sleep{(500)};
}catch(Exception e){System.out.println(e);}
System.out.println(i);
}
}
public static void main(String args[]){
TestJoinMethod1 t1=new TestJoinMethod1(1,2);
TestJoinMethod1 t2=new TestJoinMethod1(2,4,5);
TestJoinMethod1 t3=new TestJoinMethod1('sys',1,'a');
t1.run(); t1.start();
try{
t1.join();
}catch(Exception e){System.out.println(e);}
t2.start();
t3.start();
}
}
```

Correct Program

```
class TestJoinMethod2 extends Thread{
   public void run(){
       for(int i=1; i<=3; i++){
           try{Thread.sleep(500);}
           catch(Exception e){
               System.out.println(e);}
           System.out.println(i);}}

        public static void main(String args[]){
       TestJoinMethod2 t1=new TestJoinMethod2();
```

```
        TestJoinMethod2 t2=new TestJoinMethod2();
        TestJoinMethod2 t3=new TestJoinMethod2();
        t1.start();
        try{t1.join();}
        catch(Exception e){System.out.println(e);}
        t2.start();
        t3.start();
    }}
```

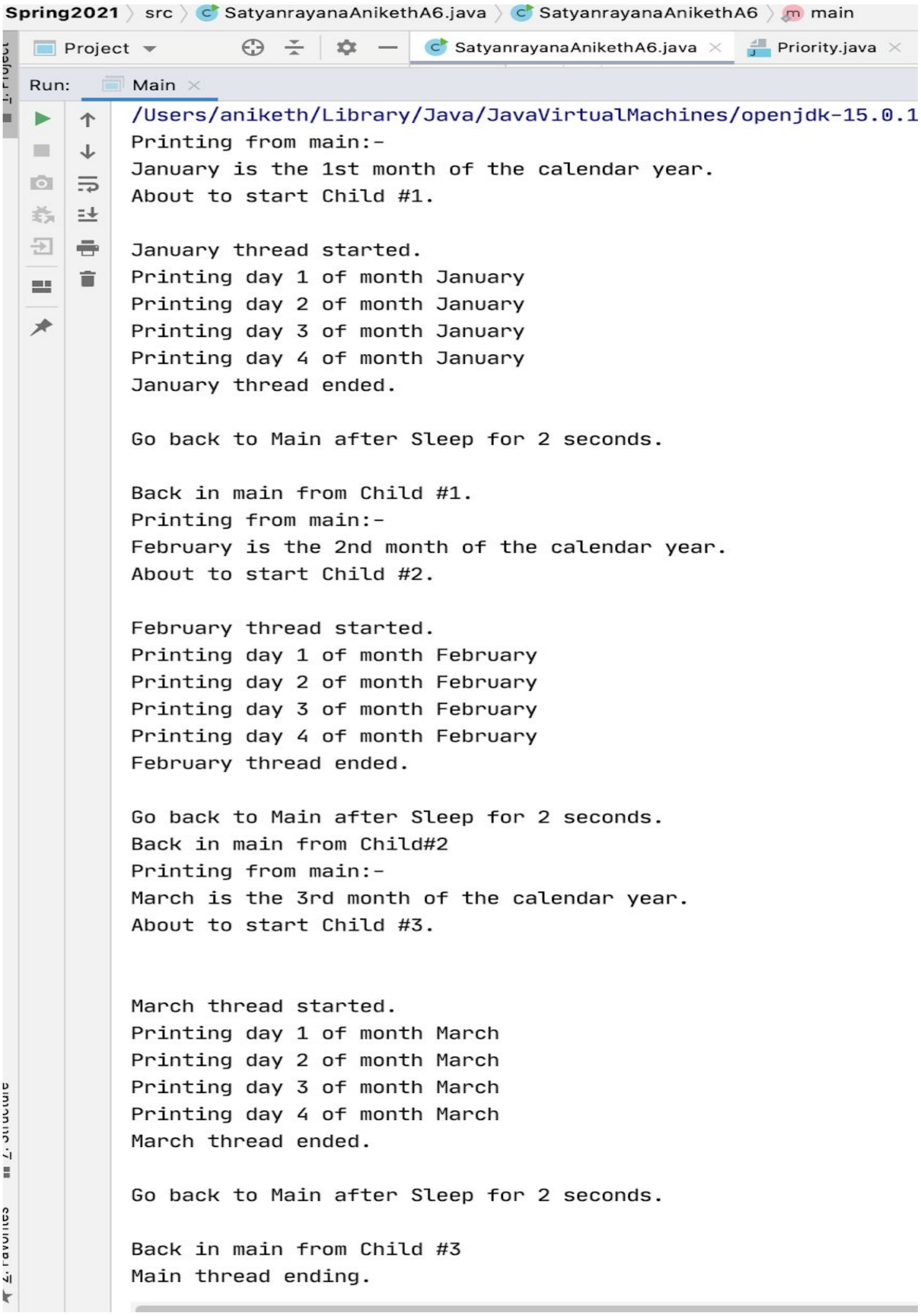
(Pasted as-is from IntelliJ IDEA)

Problem 3 : Program

```
class A6 extends Thread{
   public void run() {
        System.out.println();
        System.out.println(getName() + " thread started.");
        for (int i = 0; i < 4; i++) {
            System.out.println("Printing day " + (i + 1) +
" of month " + getName());   //getName() fetches the name
of the child thread.
        }
        System.out.println(getName() + " thread ended.");
        System.out.println();
        System.out.println("Go back to Main after Sleep for
2 seconds.");
        try {
            Thread.sleep(2000);
//child thread sleep for 2s
        } catch (InterruptedException e) {
            System.out.println("Exception");
        }
        System.out.println();
   }
}
```

```
public class SatyanrayanaAnikethA6 {
   public static void main(String[] args) {
       A6 t1 = new A6();
       A6 t2 = new A6();
       A6 t3 = new A6();

       t1.setName("January");
       t2.setName("February");
       t3.setName("March");

       System.out.println("Printing from main:-\nJanuary
is the 1st month of the calendar year.\nAbout to start
Child #1.");

       t1.start();
      try {
           t1.join(1000);
       }
       catch (InterruptedException e)
       {
           System.out.println("interrupted.");
       }
       try
       {
           Thread.sleep(2000);                     //main
thread sleep for 2s
       }catch(InterruptedException e) {
           System.out.println("Exception");
       }
       System.out.println("Back in main from Child #1.");
       System.out.println("Printing from main:-\nFebruary
is the 2nd month of the calendar year.\nAbout to start
Child #2.");

       t2.start();
       Thread.yield();
       try
```

```
        {
            Thread.sleep(2000);                 //main
thread sleep for 2s
        }catch(InterruptedException e)
        {
            System.out.println("Exception");
        }
        System.out.println("Back in main from Child#2");
        System.out.println("Printing from main:-\nMarch is
the 3rd month of the calendar year.\nAbout to start Child
#3.");

        t3.start();
        Thread.yield();
        try
        {
            Thread.sleep(2000);                 //main
thread sleep for 2s
        }catch(InterruptedException e)
        {
            System.out.println("Exception");
        }
        System.out.println("Back in main from Child #3");
        System.out.println("Main thread ending.");
    }
}
```


Spring2021 〉 src 〉 SatyanrayanaAnikethA6.java 〉 SatyanrayanaAnikethA6 〉 main
Project
SatyanrayanaAnikethA6.java
Priority.java
Run:
Main

```
/Users/aniketh/Library/Java/JavaVirtualMachines/openjdk-15.0.1
Printing from main:-
January is the 1st month of the calendar year.
About to start Child #1.

January thread started.
Printing day 1 of month January
Printing day 2 of month January
Printing day 3 of month January
Printing day 4 of month January
January thread ended.

Go back to Main after Sleep for 2 seconds.

Back in main from Child #1.
Printing from main:-
February is the 2nd month of the calendar year.
About to start Child #2.

February thread started.
Printing day 1 of month February
Printing day 2 of month February
Printing day 3 of month February
Printing day 4 of month February
February thread ended.

Go back to Main after Sleep for 2 seconds.
Back in main from Child#2
Printing from main:-
March is the 3rd month of the calendar year.
About to start Child #3.


March thread started.
Printing day 1 of month March
Printing day 2 of month March
Printing day 3 of month March
Printing day 4 of month March
March thread ended.

Go back to Main after Sleep for 2 seconds.

Back in main from Child #3
Main thread ending.
```

Note :- The lines in black are main() thread's operations.
The lines in blue are child thread's operations.

3 child threads(t1,t2,t3) are spawned from 'MAIN',
and they are named "January" , "February" and "March" respectively.

Print statement in main thread is executed.

Control is passed to the January thread(t1.start()).
t1.join(1000) -joining after 1000 seconds-main waits for t1 to finish.
(meanwhile the main thread sleeps for 2s).
Child Thread #1 starts its execution.
Print statement indicating the beginning of the January thread.
Prints first 4 days of the month January.
Print statement indicating the end of January thread.
Sleep for 2 seconds.
Child Thread #1 ends its execution.
Control is handed back to the main thread.

Print statement in main thread is executed.

Control is passed to February thread(t2.start()).
Thread.yield();
(meanwhile the main thread sleeps for 2s).
Child Thread #2 starts its execution.
Print statement indicating the beginning of Feb thread.
Prints first 4 days of the month February.
Print statement indicating the end of Feb thread.
Sleep for 2s.
Child Thread #2 ends its execution.
Control is handed back to the main thread.

Print statement in main thread is executed.

Control is passed to March thread(t3.start()).
Thread.yield();

(meanwhile the main thread sleeps for 2s).
Child Thread #3 starts its execution.
Print statement indicating the beginning of the thread.
Prints first 4 days of the month March.
Print statement indicating the end of March thread.
Sleep for 2s.
Child Thread #3 ends its execution.
Control is handed back to the main thread.
Main thread ends.

Explanation for the methods:-

1) Thread.yield(): Whenever a thread calls java.lang.Thread.yield method, it gives hint to the thread scheduler that it is ready to pause its execution.
If any thread executes the yield method, thread scheduler checks if there is any thread with same or high priority than this thread. If the processor finds any thread with higher or same priority, then it will move the current thread to Ready/Runnable state and give the processor to other thread and if not – current thread will keep executing.

2) Thread.sleep(): This method causes the currently executing thread to sleep for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.

3) Thread.currentThread().getName() gets the name of the currently-running thread.

4) The join() method of a Thread instance is used to join the start of a thread's execution to the end of other thread's execution such that a thread does not start running until another thread ends. If join() is called on a Thread instance, the currently running thread will block until the Thread instance has finished executing. The join() method waits at most this much milliseconds for this thread to die.