Java – Threading

Basics of multithreading, main thread, thread life cycle, creation of multiple threads, thread priorities, thread synchronization, inter thread communication, deadlocks for threads, suspending & resuming threads.

(Included under MAKAUT Lab Syllabus, but not specifically under Theory Syllabus!)

UD

Sep-2023

Introduction

- **A multithreaded program contains 2 or more <u>part</u>s that can run concurrently**
  - ✓ **Each such part is called a <u>Thread</u> (smallest unit of dispatchable code)**
  - ✓ **Each thread defines a separate path of execution**

- **Multithreading is specialized form of multitasking**

- **2 types of Multitasking:**
  - ✓ **Process-based (not under Java control): allows computer to run 2 or more programs concurrently, e.g, running text editor, browsing internet and playing music**

  - ✓ **Thread-based (Java feature): allows a single program to perform 2 or more tasks at once, e.g. Tasks for a text editor: (i) formatting text using text editor, (ii) printing at the same time, (iii) reading keyboard inputs at the same time**

Multithreading Benefits & Life Cycle

- **Multithreading enables in efficient programming**
  - ✓ **Allows to utilize idle time from communicating over IO devices (slower than CPU), e.g. keyboard, disk drives, or network ports**
  - ✓ **Single-core computer system: Idle CPU time is utilized & each thread receives a slice of CPU time**
  - ✓ **Multi-core computer system: Threads actually run simultaneously**

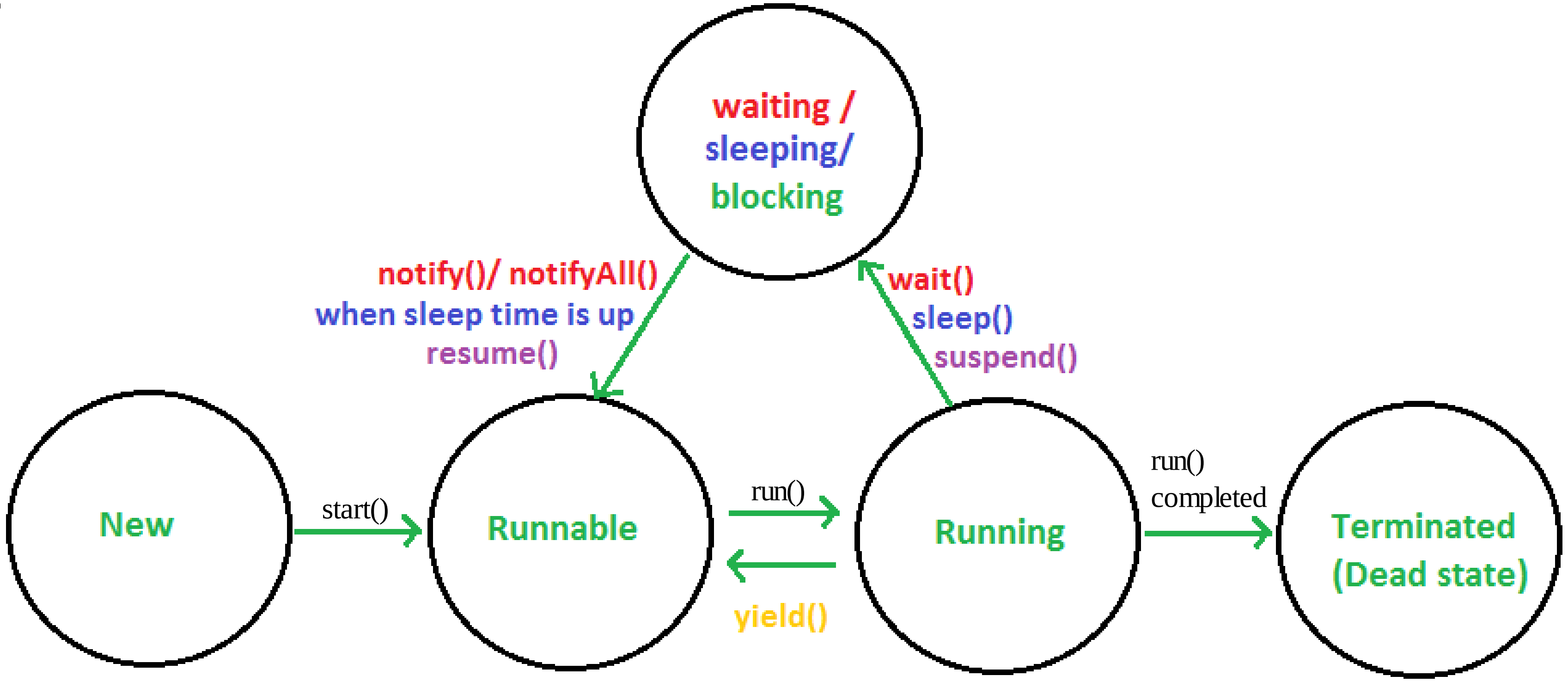

Fig. THREAD STATES

Building Thread

- **Built on Thread class encapsulating a thread of execution and / or implementing a companion interface, Runnable**

- **Both packaged under java.lang**

- **All processes have at least 1 thread of execution, main thread; executed when a Java program begins**

start()	run()
Creates a new thread and the run() method is executed on the newly created thread.	No new thread is created and the run() method is executed on the calling thread itself.
Can't be invoked more than one time otherwise throws java.lang.IllegalStateException	Multiple invocation is possible
Defined in java.lang.Thread class.	Defined in java.lang.Runnable interface and must be overriden in the implementing class.

Create Thread implementing Runnable

```
// Create thread by implementing Runnable
class MyThread implements Runnable {
    String tName;

    MyThread(String name) { tName = name; }

    public void run() {  // Entry point of a thread
       System.out.println(tName + " starting");

       try {
           for (int i=0; i<10; i++) {
              Thread.sleep(400);
              System.out.println(tName + "[" + i + "]");
           }
       }

       catch(InterruptedException exc) {
              System.out.println(tName+ " interrupted.");
       }

       System.out.println(tName + " terminating");
     }
}
```

Create Thread implementing Runnable (contd)

```
class UseThreads {
     public static void main(String[] args) {
       System.out.println("Main thread starting.");

       MyThread mt=new MyThread("C1"); //Construct a runnable object
       Thread nt=new Thread(mt); //Construct a new thread on that object
       nt.start(); // Start running the thread: runnable

       for (int i=0; i<50; i++) {
           System.out.print("M"+i+" ");
           try { Thread.sleep(100); }
           catch(InterruptedException exc) {
              System.out.println("Main thread interrupted.");
           }
       }
       System.out.println("Main thread ending.");
     }
}
```

Create Thread implementing Runnable (contd)

```
class UseThreads {
     public static void main(String[] args) {
       System.out.println("Main thread starting.");

       MyThread mt=new MyThread("C1"); //Construct a runnable object
       Thread nt=new Thread(mt); //Construct a thread on that object
       nt.start(); //Start running the thread
```

```
Output:
Main thread starting.
M0 C1 starting
M1 M2 M3 M4 C1[0]
M5 M6 M7 C1[1]
M8 M9 M10 M11 C1[2]
M12 M13 M14 M15 C1[3]
M16 M17 M18 M19 C1[4]
M20 M21 M22 M23 C1[5]
M24 M25 M26 M27 C1[6]
M28 M29 M30 M31 C1[7]
M32 M33 M34 M35 C1[8]
M36 M37 M38 M39 C1[9]
C1 terminating
M40 M41 M42 M43 M44 M45 M46 M47 M48 M49 Main thread ending.
```

```
}
```

Create Thread implementing Runnable - Improvements

```
// Create thread by implementing Runnable
class MyThread implements Runnable {
    //String tName;
    Thread thrd; //Reference to thread

    MyThread(String name) {
          // tName = name;
          thrd = new Thread(this, name); // assign thrd reference
          thrd.start(); // start thread
    }

    public void run() {  // Entry point of a thread
       // System.out.println(tName + " starting");
       System.out.println(thrd.getName() + " starting");
       try {
           for (int i=0; i<10; i++) {
              Thread.sleep(400);
              System.out.println(thrd.getName() + "[" + i + "]");
           }
       }
       catch(InterruptedException exc) {System.out.println(thrd.getName()+
" interrupted."); }
       System.out.println(thrd.getName() + " terminating");
     }
}
```

Create Thread implementing Runnable - Improvements(contd)

```
class UseThreadsImp {
     public static void main(String[] args) {
       System.out.println("Main thread starting.");
       MyThread mt=new MyThread("C1"); //Construct a runnable object
       //Thread nt=new Thread(mt); //Construct a thread on that object
       //nt.start(); //Start running the thread
       for (int i=0; i<50; i++) {
           System.out.print("M"+i+" ");
           try { Thread.sleep(100); }
           catch(InterruptedException exc) { System.out.println("Main
thread interrupted."); }
       }
       System.out.println("Main thread ending.");
     }
}
```

Create Thread implementing Runnable - Improvements(contd)

```
class UseThreadsImp {
     public static void main(String[] args) {
       System.out.println("Main thread starting.");
       MyThread mt=new MyThread("C1"); //Construct a runnable object
       //Thread nt=new Thread(mt); //Construct a thread on that object
       //nt.start(); //Start running the thread
       for (int i=0; i<50; i++) {
           System.out.print("M"+i+" ");
           try { Thread.sleep(100); }
           catch(InterruptedException exc) { System.out.println("Main
thread interrupted."); }
       }
       System.out.println("Main thread ending.");
     }
}
```

```
Output:
Main thread starting.
M0 C1 starting
M1 M2 M3 M4 C1[0]
M5 M6 M7 C1[1]
M8 M9 M10 M11 C1[2]
M12 M13 M14 M15 C1[3]
M16 M17 M18 M19 C1[4]
M20 M21 M22 M23 C1[5]
M24 M25 M26 M27 C1[6]
M28 M29 M30 M31 C1[7]
M32 M33 M34 M35 C1[8]
M36 M37 M38 M39 C1[9]
C1 terminating
M40 M41 M42 M43 M44 M45 M46 M47 M48 M49 Main thread ending.
```

Create Thread extending Thread

```
// Create thread by Thread extension. Question: Out of the 2 ways of thread
// creation, runnable is preferred over Thread extension.. Why?
class MyThread extends Thread {
    MyThread(String name) {
          super(name); // name thread
          start(); // start thread
    }
    public void run() {  // Entry point of new thread
       System.out.println(getName() + " starting");
       try {
           for (int i=0; i<10; i++) {
              Thread.sleep(400);
              System.out.println(getName() + "[" + i + "]");
           }
       }
       catch(InterruptedException exc) {System.out.println(getName()+ "
interrupted."); }
       System.out.println(getName() + " terminating");
     }
}
```

Create Thread extending Thread (contd)

```
class ExtendThread {
    public static void main(String[] args) {
       System.out.println("Main thread starting.");
       MyThread mt=new MyThread("C1"); //Construct a runnable object
       for (int i=0; i<50; i++) {
           System.out.print("M"+i+" ");
           try { Thread.sleep(100); }
           catch(InterruptedException exc) { System.out.println("Main
thread interrupted."); }
       }
       System.out.println("Main thread ending.");
    }
}
```

Create Thread extending Thread (contd)

```
class ExtendThread {
    public static void main(String[] args) {
        System.out.println("Main thread starting.");
        MyThread mt=new MyThread("C1"); //Construct a runnable object
        for (int i=0; i<50; i++) {
            System.out.print("M"+i+" ");
            try { Thread.sleep(100); }
            catch(InterruptedException exc) { System.out.println("Main
th
}
```

```
Output:
Main thread starting.
M0 C1 starting
M1 M2 M3 M4 C1[0]
M5 M6 M7 C1[1]
M8 M9 M10 M11 C1[2]
M12 M13 M14 M15 C1[3]
M16 M17 M18 M19 C1[4]
M20 M21 M22 M23 C1[5]
M24 M25 M26 M27 C1[6]
M28 M29 M30 M31 C1[7]
M32 M33 M34 M35 C1[8]
M36 M37 M38 M39 C1[9]
C1 terminating
M40 M41 M42 M43 M44 M45 M46 M47 M48 M49 Main thread ending.
```

Creating Multiple Threads

```
// Create multiple threads by implementing Runnable
class MyThread implements Runnable {
    Thread thrd; //Reference to thread

    MyThread(String name) {
          // tName = name;
          thrd = new Thread(this, name); // assign thrd reference
          thrd.start(); // start thread
    }

    public void run() {  // Entry point of a thread
       System.out.println(thrd.getName() + " starting");
       try {
           for (int i=0; i<10; i++) {
               Thread.sleep(400);
               System.out.println(thrd.getName() + "[" + i + "]");
           }
       }
       catch(InterruptedException exc) {
               System.out.println(thrd.getName()+ " interrupted."); }
       System.out.println(thrd.getName() + " terminating");
     }
}
```

Creating Multiple Threads (contd.)

```
class MultiThreads {
     public static void main(String[] args) {
               System.out.println("Main thread starting.");
               MyThread mt1=new MyThread("C1"); //Construct runnable obj.
               MyThread mt2=new MyThread("C2"); //Construct runnable obj.
               MyThread mt3=new MyThread("C3"); //Construct runnable obj.
               for (int i=0; i<50; i++) {
                   System.out.print("M"+i+" ");
                   try { Thread.sleep(100); }
                 catch(InterruptedException exc) {
                  System.out.println("Main thread interrupted." ); }
        }
        System.out.println("Main thread ending.");
     }
}
```

Creating Multiple Threads (contd.)

```
class MultiThreads {
     public static void main(Strin
               System.out.println(
               MyThread mt1=new My
               MyThread mt2=new My
               MyThread mt3=new My
               for (int i=0; i<50;
                   System.out.prin
                   try { Thread.sl
                catch(InterruptedE
                  System.out.print
        }
        System.out.println("Main th
     }
}
```

```
Output:
Main thread starting.
C1 starting
C2 starting
M0 C3 starting
M1 M2 M3 C1[0]
C3[0]
C2[0]
M4 M5 M6 M7 C1[1]
C3[1]
C2[1]
M8 M9 M10 M11 C1[2]
C3[2]
C2[2]
M12 M13 M14 M15 C1[3]
C3[3]
C2[3]
M16 M17 M18 M19 C1[4]
C3[4]
C2[4]
M20 M21 M22 M23 C1[5]
C3[5]
C2[5]
M24 M25 M26 M27 C1[6]
C3[6]
C2[6]
M28 M29 M30 M31 C1[7]
C3[7]
C2[7]
M32 M33 M34 M35 C1[8]
C3[8]
C2[8]
M36 M37 M38 M39 C1[9]
C1 terminating
C3[9]
C3 terminating
C2[9]
C2 terminating
M40 M41 M42 M43 M44 M45 M46 M47 M48 M49 Main thread ending.
```

Determining when a thread ends

Q: Refer previous examples. Why did we use calculated sleep() in the main thread?
A: The main thread should not complete before the child threads complete their executions.

Q: So.. we need to calculate execution times of all the child threads, and put appropriate sleep() in the main thread.. Isn't there a better way?
A: Sure.. Main thread simply needs to wait as long as the child threads are alive.. 2 ways of checking:

(i) isAlive()

- **Method mt1.thrd.isAlive() returns true as longs as C1 is alive**
- **Check within a loop under main thread and terminate when false**
- **Add sleep() to allow switching of threads**

(ii) join()

- **mt1.thrd.join() will force main thread to wait until C1 ends and joins the main thread**

Multithreading using join()

```
// Create multiple threads by implementing Runnable.. MyThread as before
class MyThread implements Runnable {
    Thread thrd; // Reference to thread

    MyThread(String name) {
           // tName = name;
           thrd = new Thread(this, name); // assign thrd reference
           thrd.start(); // start thread
    }

    public void run() {  // Entry point of a thread
        System.out.println(thrd.getName() + " starting");
        try {
            for (int i=0; i<10; i++) {
                Thread.sleep(400);
                System.out.println(thrd.getName() + "[" + i + "]");
            }
        }
        catch(InterruptedException exc) {
                System.out.println(thrd.getName()+ " interrupted."); }
        System.out.println(thrd.getName() + " terminating");
     }
}

// Visible changes now under main thread
```

Multithreading using join().. Contd.

```java
class JoinThreads {
    public static void main(String[] args) {
        System.out.println("Main thread starting.");
        MyThread mt1=new MyThread("C1"); // Construct runnable object
        MyThread mt2=new MyThread("C2"); // Construct runnable object
        MyThread mt3=new MyThread("C3"); // Construct runnable object

        try {
            mt1.thrd.join();
            System.out.println("C1 joined after thread completion");
            mt2.thrd.join();
            System.out.println("C2 joined after thread completion");
            mt3.thrd.join();
            System.out.println("C3 joined after thread completion");
        }
        catch(InterruptedException exc) {
            System.out.println("Main thread interrupted." ); }

        System.out.println("Main thread ending.");
    }
}
```

Multithreading using joi

```
class JoinThreads {
    public static void main(String[] args) {
            System.out.println("Main thread st
            MyThread mt1=new MyThread("C1"); /
            MyThread mt2=new MyThread("C2"); /
            MyThread mt3=new MyThread("C3"); /

            try {
                  mt1.thrd.join();
                  System.out.println("C1 join
                  mt2.thrd.join();
                  System.out.println("C2 join
                  mt3.thrd.join();
                  System.out.println("C3 join
            }
            catch(InterruptedException exc) {
                  System.out.println("Main th

            System.out.println("Main thread en
    }
}
```

```
Output: Main thread starting.
C1 starting
C3 starting
C2 starting
C1[0]
C3[0]
C2[0]
C1[1]
C3[1]
C2[1]
C1[2]
C3[2]
C2[2]
C1[3]
C3[3]
C2[3]
C1[4]
C3[4]
C2[4]
C1[5]
C3[5]
C2[5]
C1[6]
C3[6]
C2[6]
C1[7]
C3[7]
C2[7]
C1[8]
C3[8]
C2[8]
C1[9]
C1 terminating
C3[9]
C1 joined after thread completion
C2[9]
C3 terminating
C2 terminating
C2 joined after thread completion
C3 joined after thread completion
Main thread ending.
```

Thread priorities

- Generally High-priority (HPR) thread has higher access to CPU compared to Lower-priority (LPR) threads

- However.. If HPR thread is waiting on resource (e.g. keyboard input), then it will be blocked and a LPR thread will run.. except in non-preemptive operating system..

- When child thread starts, its priority is same as parent thread

- Above can be changed, e.g. final void setPriority(int *level*)
  - ✓ Value of level: (MIN_PRIORITY..MAX_PRIORITY), i.e. 1 to 10
  - ✓ Default Priority: NORM_PRIORITY, i.e. 5

- Obtain current priority: final int getPriority()

- For managing threads interaction, Java Synchronization should be used instead of priority setting

Synchronization

- **Coordinating tasks of 2 or more threads when accessing a shared resource, e.g. file writing, resuming a thread in suspended state etc.**

- **All Java objects support synchronization, leveraging the concept of monitor -> lock**

- **Use synchronized keyword: (i) for synchronizing a method or (ii) for synchronizing a block**

- **No separate programming will be needed. Consider the sequences:**

  - ✓ **synchronized method called -> calling thread enters object's monitor -> object locked and no other thread can enter method on that object**

  - ✓ **Thread returns from synchronized method -> monitor unlocks object allowing next thread to use**

Synchronization (contd)

```
// Synchronize access to method for adding array elements
class SumArrayClass {            // Sum logic
      private int sum;
      synchronized int sumArray( int[] nums) {
            sum=0;
            for(int i=0; i<nums.length; i++) {
                  sum += nums[i]; // cumulative sum

      System.out.println(Thread.currentThread().getName() +
"/Added: " + nums[i] + "/Total: " + sum );

                  try { Thread.sleep(10); } // Allow task switch
                  catch(InterruptedException exc) {
System.out.println( "Thread interrupted"); }
            }

            return sum;
      }
}
```

Synchronization (contd)

```
class MyThread implements Runnable {    // Thread manipulation
      Thread thrd; //Reference to thread
      static SumArrayClass sa = new SumArrayClass(); // sum
logic
      int result;
      int[] temp;
      MyThread(String name, int[] nums) { // Construct new thrd
            thrd = new Thread(this, name);
         temp = nums;
            thrd.start(); // start thread
      }
      public void run() {  // Start execution of new thread
            int sum;
            System.out.println(thrd.getName() + " starting");
            result = sa.sumArray( temp );
            System.out.println(thrd.getName() + "/Sum: " +
result );
            System.out.println(thrd.getName() + " terminating");
     }
}
```

Synchronization (contd)

```
class SumArraySync {        // Main class
     public static void main(String[] args) {
            int[] arr = { 1, 2, 3, 4, 5 /** .., , ,.. **/ };
            MyThread mt1=new MyThread("C1", arr);
            MyThread mt2=new MyThread("C2", arr);

            try {
                  mt1.thrd.join();
                  System.out.println("C1 joined..");

                  mt2.thrd.join();
                  System.out.println("C2 joined..");

            }
           catch(InterruptedException exc) {
System.out.println("Main thread interrupted" );
      }
      }
}
```

Synchronization (contd)

```
class SumArraySync {
     public static void main(String[] args) {
            int[] arr = { 1, 2, 3, 4, 5 };
            MyThread mt1=new MyTh
            MyThread mt2=new MyTh

            try {
                  mt1.thrd.join()
                  System.out.prin
                  mt2.thrd.join()
                  System.out.prin
            }
           catch(InterruptedExcep
System.out.println("Main thread i
      }
}
```

```
Output:
C1 starting
C2 starting
C1/Added: 1/Total: 1
C1/Added: 2/Total: 3
C1/Added: 3/Total: 6
C1/Added: 4/Total: 10
C1/Added: 5/Total: 15
C1/Sum: 15
C2/Added: 1/Total: 1
C1 terminating
C1 joined..
C2/Added: 2/Total: 3
C2/Added: 3/Total: 6
C2/Added: 4/Total: 10
C2/Added: 5/Total: 15
C2/Sum: 15
C2 terminating
C2 joined..
```

Synchronization (contd)

Q: What will happen if synchronized keyword is removed from the previous program?

A: **sumArray()** no longer synchronized and any number of threads can execute it concurrently. So, running total in **sum** will be changed by each thread through the static object sa. This will mix together summation of both threads producing incorrect result.

This is called race condition, and happened in the absence of proper synchronization of threads.

Synchronization (contd)

Q: What will happen if synchronized keyword is removed from the previous program?

A: **sumArray()** no longer synchronized and any number of threads
can execute it concurrently. So, r
changed by each thread through the
mix together summation of both thr
result.

This is called race condition, and
proper synchronization of threads.

```
Output:
C2 starting
C1 starting
C2/Added: 1/Total: 1
C1/Added: 1/Total: 1
C2/Added: 2/Total: 3
C1/Added: 2/Total: 5
C2/Added: 3/Total: 8
C1/Added: 3/Total: 11
C2/Added: 4/Total: 15
C1/Added: 4/Total: 19
C2/Added: 5/Total: 24
C1/Added: 5/Total: 29
C2/Sum: 29
C1/Sum: 29
C2 terminating
C1 terminating
C1 joined..
C2 joined..
```

Synchronized block

- Q: Source code for 3rd party method may not be available. How can we apply **synchronized** statement?
- A: Call to this method should be put under a **synchronized** block, e.g.

```
synchronized(sa){ // Calls to sumArray() on sa are synchronized
      result = sa.sumArray( temp );
}
```

- Obviously method declaration could not use **synchronized, e.g.**

```
/* synchronized */ int sumArray( int[] nums) {
```

- Output will remain the same

Additionals: Interthread Communication & Deadlock

- Methods are **wait()**, **notify()** and notifyAll()
- Above implemented by **Object** class, and part of all Java objects
- **wait()**: Calling thread releases control of the object and suspends until it receives a notification, e.g.

```
    final void wait() throws InterruptedException
    final void wait(long mills) throws InterruptedException
    final void wait(long mills, int nanos) throws
InterruptedException
```

- **notify()**: Resumes one waiting thread, e.g.

```
    final void notify()
```

- **notifyAll()**: Notifies all threads and highest priority thread gains access to the object, e.g.

```
    final void notifyAll()
```

- Deadlock arises when Thread 1 is waiting on Thread 2 for accessing a shared object (resource), but.. Thread 2 is already waiting on Thread 1 -> Neither executes -> program hangs
- **Solution to deadlock:** Prevention with proper synchronization of threads

Suspending, Resuming & Stopping Threads

- **Prior to Java 2**, methods were **suspend()** to pause, **resume()** to restart and **stop()** to stop execution of thread
- Above methods no longer used as serious problems were found incl. deadlock specially from **suspend()** method
- **Solution?** Design thread so that **run()** method periodically checks to decide whether to suspend, resume or stop itself

- **Design:**
  - ✓ **boolean suspended**: run() method must continue for thread execution; use wait() in a loop when the flag is set
  - ✓ **boolean stopped**: stop execution when set; use break from while loop
  - ✓ Constructor to initialize flags as false. **run()** method to use **synchronized** statement for checking **suspended** & **stopped** flags
  - ✓ Use **synchronized** Methods:

    mySuspend() {suspended=true}: for suspending thread execn
    myResume {suspended=false; notify()}: for resume execution
    myStop {stopped=true; suspened=false; notify()}: to stop