Multithreading

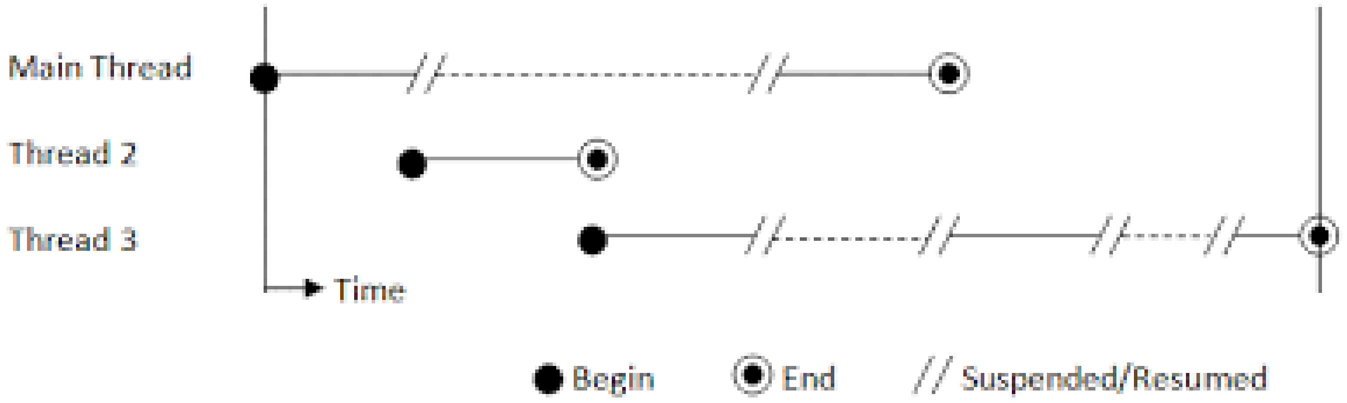
REVATURE

Topics

1) Multithreading
2) Lifecycle of a Thread
3) Priorities
4) Creating a Thread
5) Runnable interface VS Thread Class
6) Thread methods
7) Synchronization
8) Deadlock

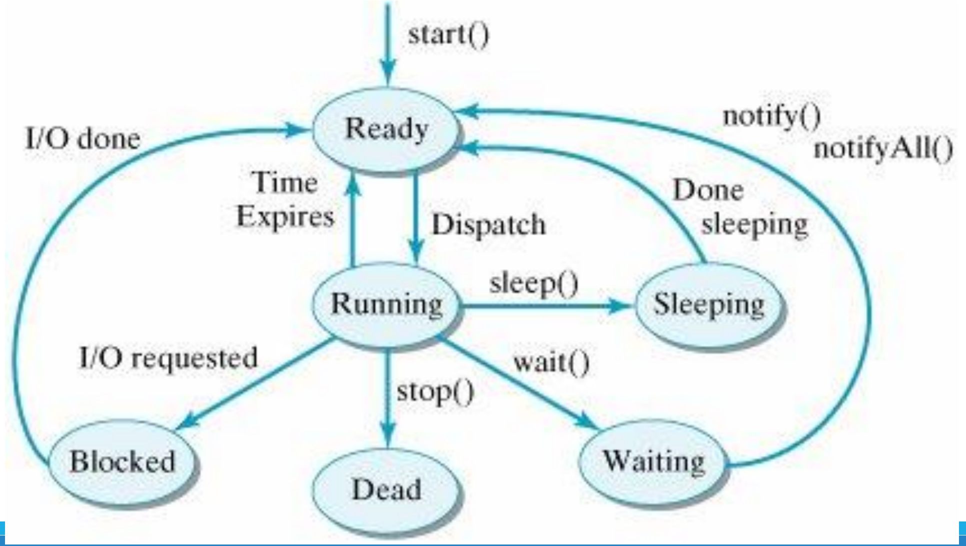
Multithreading

- A multithreaded program can have two or more threads running concurrently
- Each thread can have its own task
- Program becomes optimized and runs faster
- OS divides processing time not just with applications but between threads as well



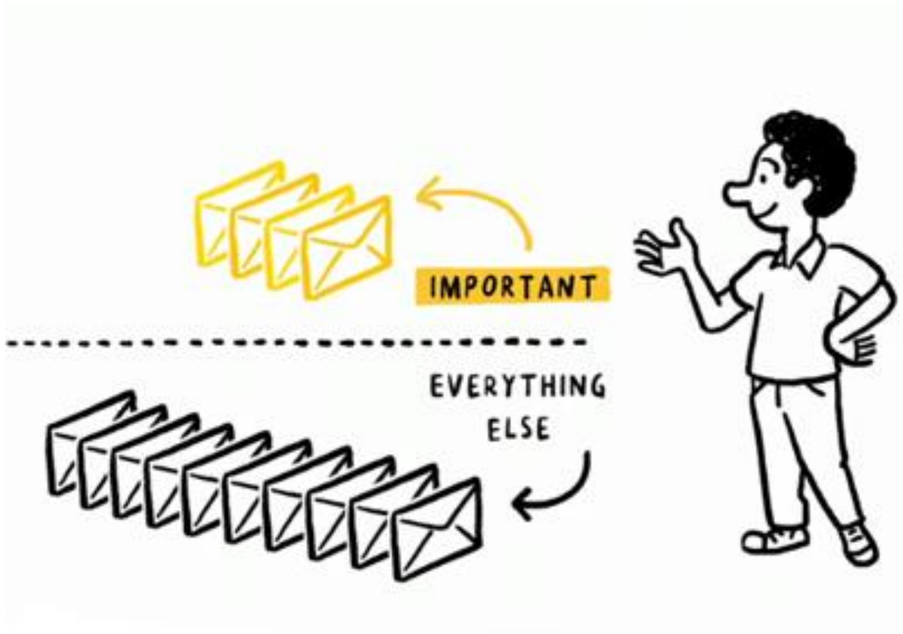
Lifecycle of a Thread

- New: A thread’s lifecycle begins. Stays in this state until the program starts the thread.
- Runnable: After the thread starts, its in the runnable state.
- Waiting: Waits for other threads to complete the task. Will go back to runnable state only when the other thread signals it to continue executing.
- Timed Waiting: A thread can be in a waiting state for a specified interval time. It goes back to the runnable state after the time has expired.
- Terminated: A thread that completes the task is terminated.



Priorities

- Threads can be configured with a priority number which signifies which order threads are to be run in (Range is 1 – 10).
- MIN_PRIORITY (typically a 1)
- NORM_PRIORITY (defaults to 5)
- MAX_PRIORITY (typically a 10)

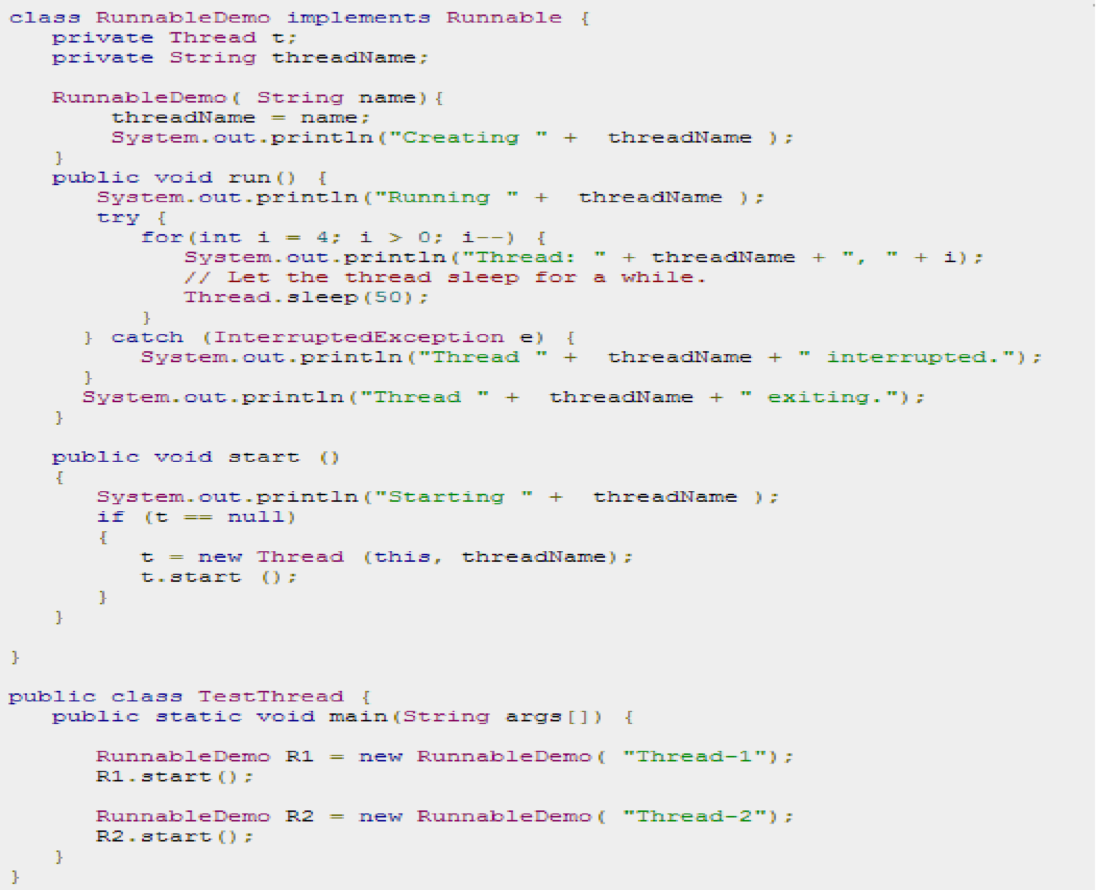


Creating a Thread

- Create a class that **implements** the Runnable interface
  - Implement the run() method
  - Pass an object of it into the Thread constructor
  - Call the start() method
- Create a class that **extends** the Thread class
  - Override the run() method
  - Create an object of the class
  - Call the start() method

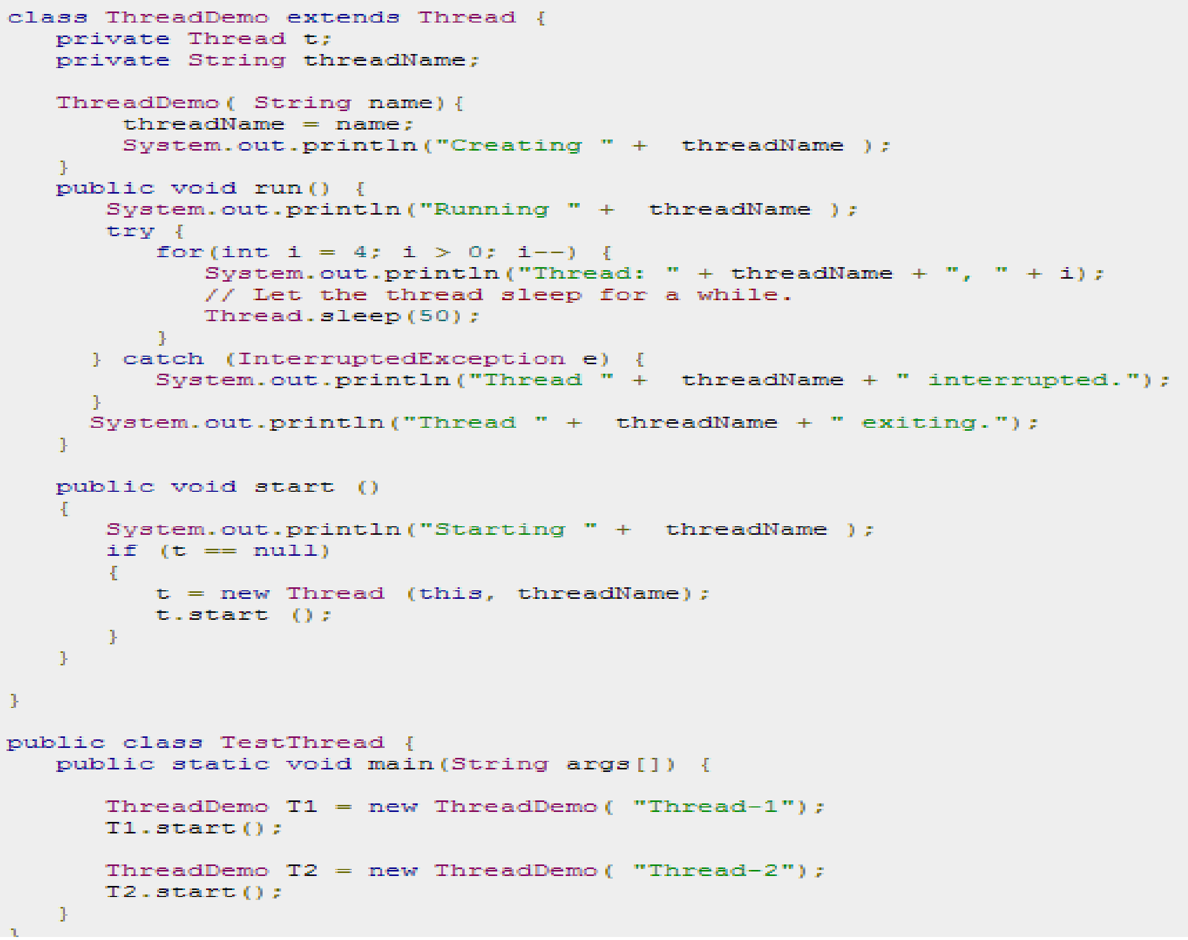
Thread	Runnable
When you want to override other Thread utility methods	When you want to extend another class

Implementing Runnable



```java
class RunnableDemo implements Runnable {
   private Thread t;
   private String threadName;

   RunnableDemo( String name){
       threadName = name;
       System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
     } catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
     }
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}

public class TestThread {
   public static void main(String args[]) {

      RunnableDemo R1 = new RunnableDemo( "Thread-1");
      R1.start();

      RunnableDemo R2 = new RunnableDemo( "Thread-2");
      R2.start();
   }
}
```

Extending Thread



```
class ThreadDemo extends Thread {
   private Thread t;
   private String threadName;

   ThreadDemo( String name){
       threadName = name;
       System.out.println("Creating " +  threadName );
   }
   public void run() {
      System.out.println("Running " +  threadName );
      try {
         for(int i = 4; i > 0; i--) {
            System.out.println("Thread: " + threadName + ", " + i);
            // Let the thread sleep for a while.
            Thread.sleep(50);
         }
     } catch (InterruptedException e) {
         System.out.println("Thread " +  threadName + " interrupted.");
     }
     System.out.println("Thread " +  threadName + " exiting.");
   }

   public void start ()
   {
      System.out.println("Starting " +  threadName );
      if (t == null)
      {
         t = new Thread (this, threadName);
         t.start ();
      }
   }

}

public class TestThread {
   public static void main(String args[]) {

      ThreadDemo T1 = new ThreadDemo( "Thread-1");
      T1.start();

      ThreadDemo T2 = new ThreadDemo( "Thread-2");
      T2.start();
   }
}
```

Thread Methods

- Non Static methods
  - public void start()
  - public void run()
  - public final void setName(String name)
  - public final void setPriority(int priority)
  - public final void setDaemon(boolean on)
  - public final void join(long millisecond)
  - public void interrupt()
  - public final boolean isAlive()

Thread Methods

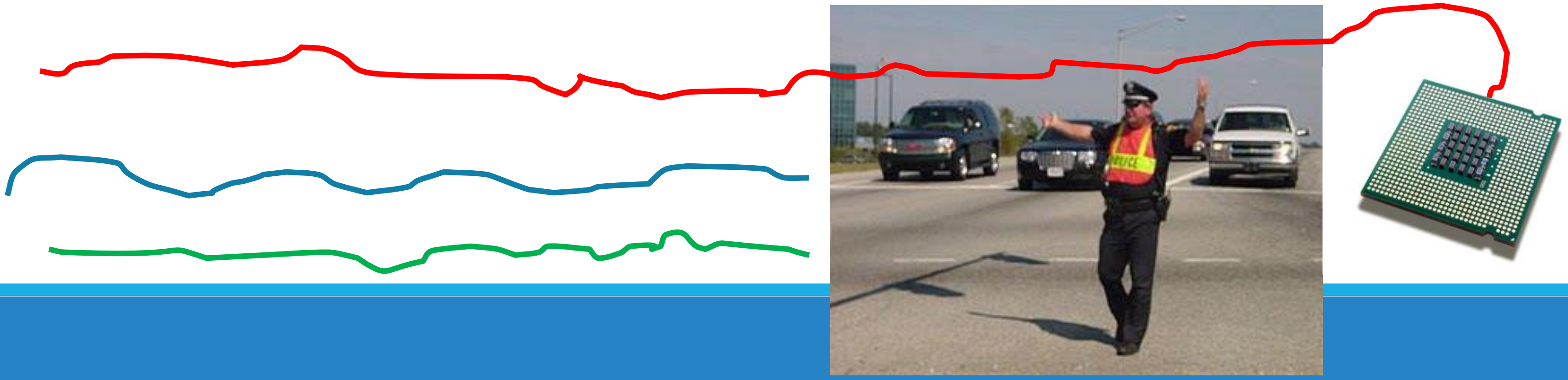
- Static method
  - public static void yield()
  - public static void sleep(long milliseconds)
  - public static boolean holdsLock(Object x)
  - public static Thread currentThread()
  - public static void dumpStack()

Thread Methods

- Object methods that threads can use
  - notify()
  - notifyAll()
  - wait()

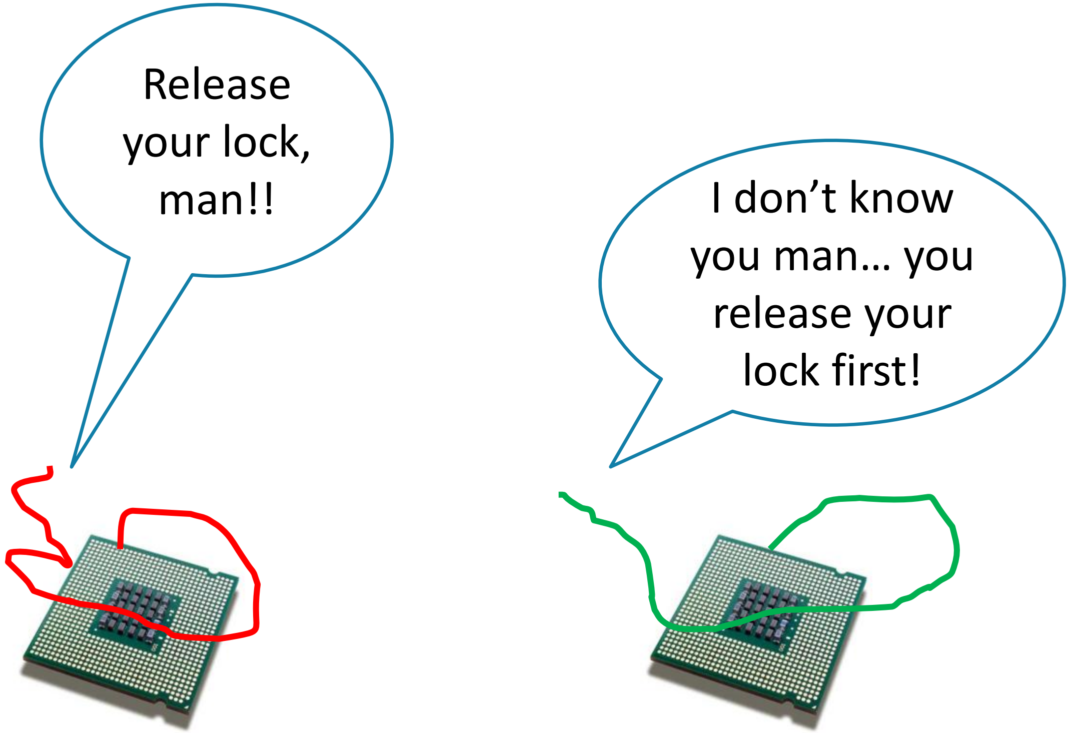
Synchronization

- Problem: When two or more threads are trying to access a method or process, a race condition may occur. If one thread is writing data to a file, the other thread may overwrite that data which may cause issues in the program.
- Solution: synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called *monitors*. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.
- The Java programming language provides a very handy way of creating threads and synchronizing their task by using **synchronized** blocks. You keep shared resources within this block.



Deadlock

- Occurs due to resource locking
- Thread ‘A’ locks resource ‘1’
- Thread ‘B’ locks resource ‘2’
- Thread ‘A’ requests resource ‘2’
- Thread ‘B’ requests resource ‘1’
- Neither thread can continue!
- Each holds locks on resources the other needs



Review

1) Multithreading
2) Lifecycle of a Thread
3) Priorities
4) Creating a Thread
5) Runnable interface VS Thread Class
6) Thread methods
7) Synchronization
8) Deadlock

Assignment

- Write a program that creates a thread using either a Runnable interface or Thread class. Have it call a method which prints out numbers from 1 to 10. Hint: Loop the thread.
- Write a program which demonstrates synchronization between two threads. Print out the duration a thread has to wait till the prior one has completed. Loop the threads 5 times each.
- Review deadlock information in your book or online.