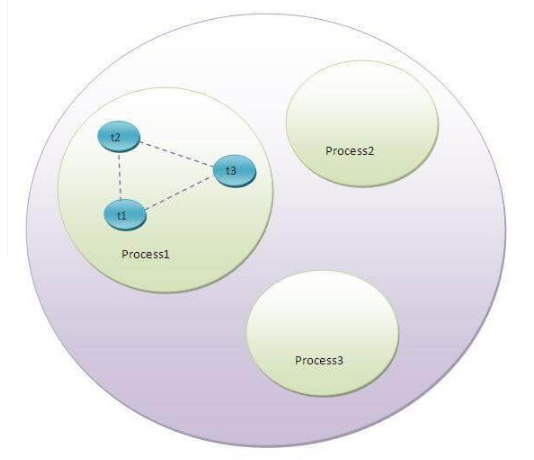
**CSE 310**

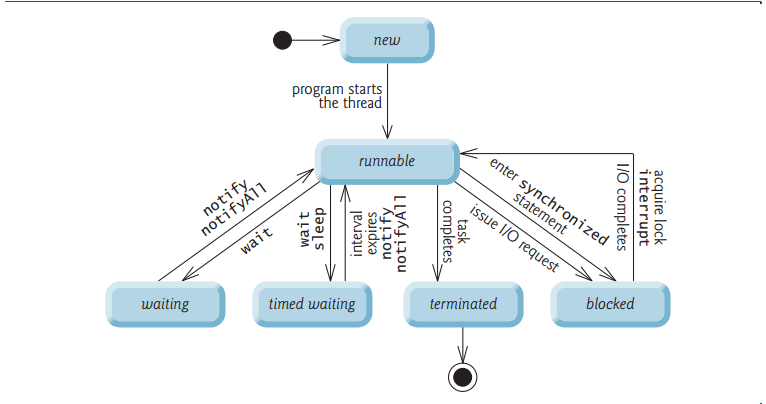
**Week 4**

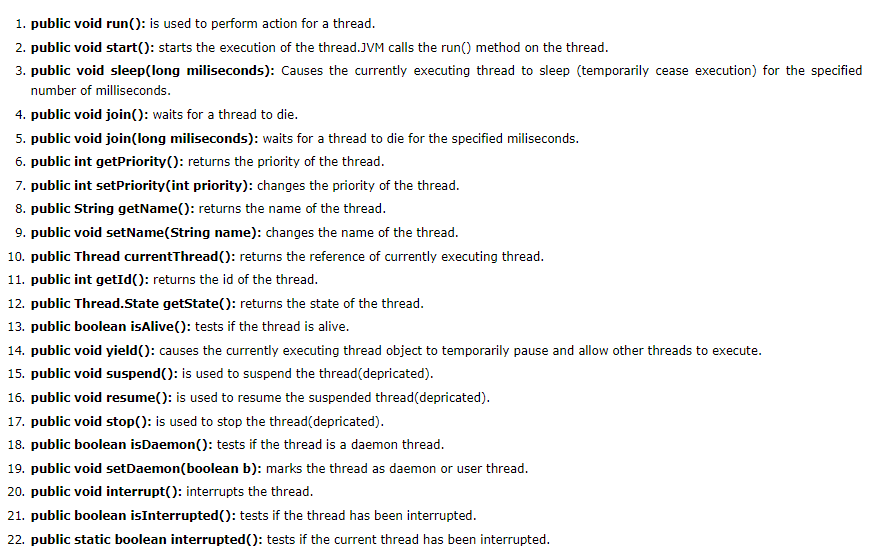
1. **Thread:**
   1. Athread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.
   2. Threads are independent, if there occurs exception in one thread, it doesn't affect other threads.
   3. It shares a common memory area.

****

* 1. Context switching.

1. **Multitasking:**
   1. Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:
      1. Process-based Multitasking(Multiprocessing)
      2. Thread-based Multitasking (Multithreading)
   2. **Process-based multitasking**
      1. Each process have its own address in memory i.e. each process allocates separate memory area.
      2. Process is heavyweight.
      3. Cost of communication between the processes is high.
      4. Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.
   3. **Thread-based multitasking**
      1. Threads share the same address space.
      2. Thread is lightweight.
      3. Cost of communication between the thread is low.
2. **Lifecycle of a thread:**
   * 1. The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:
        1. New
        2. Runnable
        3. Waiting
        4. Timed Waiting
        5. Blocked
        6. Terminated



1. **Creating thread:**
   1. There are two ways to create a thread:
      1. By extending Thread class
      2. By implementing Runnable interface
   2. **Creating thread by extending Thread class:**
      1. Thread class:
         1. Thread class provide constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.
      2. **Commonly used constructors of Thread class:**
         1. Thread()
         2. Thread(String name)
         3. Thread(Runnable r)
         4. Thread(Runnable r, String name)
      3. **Commonly used methods of Thread class:**
      4. Please refer to codes: **threadExtendingThreadClass.java** file for example.
   3. **Creating threads by implementing Runnable interface:**
      1. The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().
      2. **public void run():**is used to perform action for a thread.
      3. **Please refer to code:** **threadImplementingRunnable.java** file for example.
      4. If you are not extending the Thread class, your class object would not be treated as a thread object. So you need to explicitly create Thread class object. We are passing the object of your class that implements Runnable so that your class run() method may execute.
   4. **Sleeping thread:**
      1. **Sleep method in java:**
         1. The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
         2. The Thread class provides two methods for sleeping a thread:
            1. public static void sleep(long miliseconds)throws InterruptedException
            2. public static void sleep(long miliseconds, int nanos)throws InterruptedException
      2. As you know well that at a time only one thread is executed.If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on.
      3. Please refer to code: **TestSleepMethod1.java**
   5. **Can we start a thread twice?**
      1. No. After starting a thread, it can never be started again. If you does so, an IllegalThreadStateException is thrown. In such case, thread will run once but for second time, it will throw exception.
      2. Please refer to code: **TestThreadTwice1.java**
   6. **What if we call run() method directly instead start() method?**
      1. Each thread starts in a separate call stack.
      2. Invoking the run() method from main thread, the run() method goes onto the current call stack rather than at the beginning of a new call stack.

**class** TestCallRun1 **extends** Thread{

**public** **void** run(){

 System.out.println("running...");

}

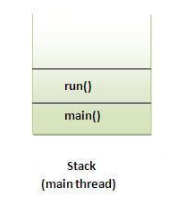
**public** **static** **void** main(String args[]){

  TestCallRun1 t1=**new** TestCallRun1();

  t1.run();//fine, but does not start a separate call stack

 }

}



* + 1. Please refer to code: **directCallRunMethod.java**
    2. No context switching. Please refer to code: **directCallRunContextSwitch.java**