Pemrograman Jaringan 7

anton@ukdw.ac.id

Thread

- Java menyediakan fasilitas pemgrograman Thread bahkan multithreading.
- Thread memperbolehkan beberapa aktivitas berjalan bersamaan dalam satu program yang sama.
- Free Online Dictionary of Computing (FOLDOC):

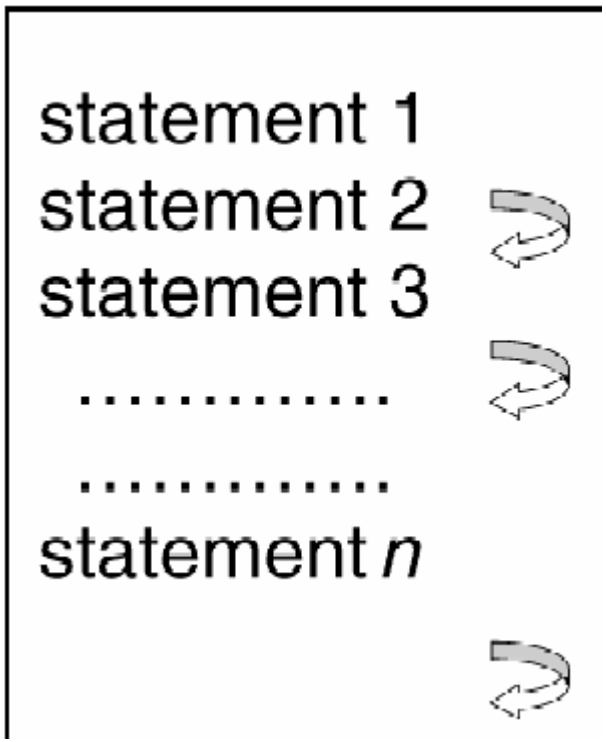
  Sharing a single CPU between multiple tasks (or "threads") in a way designed to minimize the time required to switch tasks. This is accomplished by sharing as much as possible of the program execution environment between the different tasks so that very little state needs to be saved and restored when changing tasks.

Processes vs. Threads

- Processes
  - Completely separate, unrelated concurrent execution on the level of the operating system. (eg multiple programs running at the same time)
- Threads
  - Concurrent units of execution within a given program. (eg pulling down a menu while loading a web page within a web browser)

Single Thread

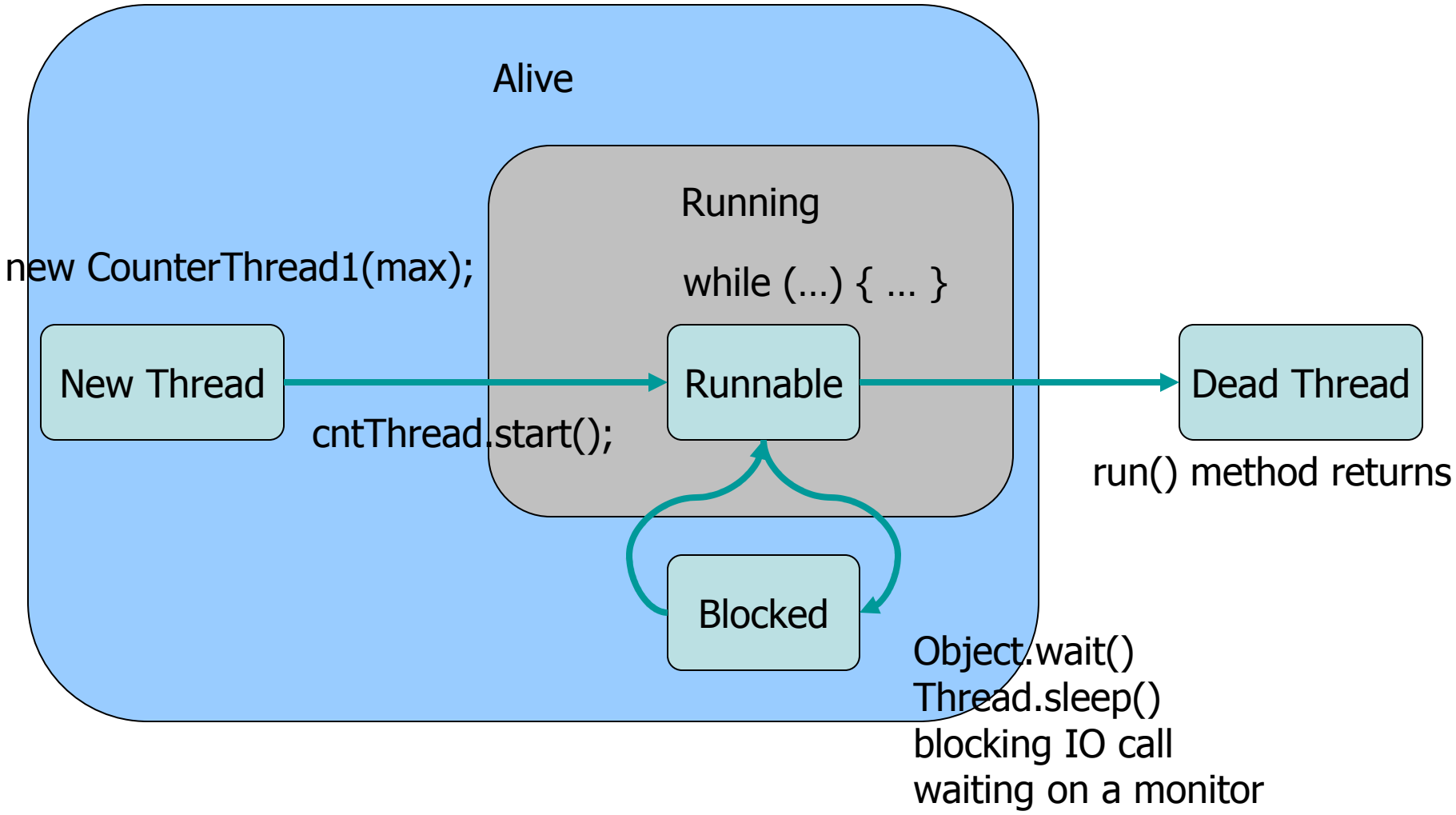
- In single-threaded execution, statements are executed sequentially.



Contoh Theads

- a GUI-driven program may be displaying a background animation while processing the user's
- foreground interactions with the interface, or a Web browser may need to download and display the contents of a graphics file while rendering the rest of the associated Web page.
- Windows vs DOS
- Desktop Email Reader : kita dapat mendownload email sambil membaca email. Program ini tidak harus menunggu semua email kita download semua dahulu, melainkan untuk satu email yang telah kita download dapat langsung dibaca.
- Word Processor : kita dapat mengeprint sambil mengetik. Program ini juga mampu mengecek spelling dan grammar sambil kita mengetik.
- JVM dari java juga menggunakan Thread untuk Garbage Collector nya, dimana Garbage Collector akan menghapus semua memori yang sudah dipakai program java yang sudah tidak digunakan lagi.

Thread State Diagram



Thread State

- **new**: The thread is being created
- **running**: Instructions are being executed
- **waiting**: The thread is waiting for some event to occur
- **ready**: The thread is waiting to be assigned to a processor
- **terminated**: The thread has finished execution

Life cycle of a Thread (cont'd)

- The OS can interrupt the thread at **any** time while it is running, and allow **any** other thread to run.
- Threads can put themselves into a wait state until another thread wakes them up.

What can go wrong?

- Assuming that threads, existing in the same program, have access to the same variable.
- What if one is reading data from an array, gets interrupted, and another one writes to that array, even though the thread wanted the old values?
- Must be **Synchronized**!

How does Java handle Threads?

- Packages java.lang.Thread
- Subclass java.lang.Thread, or implement java.lang.Runnable.
- After you instantiate a thread, it is in the *ready* state.
- To start running a Thread, call the start method on it.

How does Java handle them? (cont'd)

- To implement Runnable, you need to have a method that override

  ```
  public void run();
  ```

- This is the method that implements the running section of the thread life-cycle.
- The thread dies (stops) when the run method terminates.
- The run method may be interrupted at any time by the operating system and put into the interrupted state, but that's not something you really need to handle.

Thread class

Table 1-1 Constructors and Methods of the Thread Class	
Constructor	***Explanation***
`Thread()`	The basic `Thread` constructor without parameters. This constructor simply creates an instance of the `Thread` class.
`Thread(String name)`	Creates a `Thread` object and assigns the specified name to the thread.
`Thread(Runnable target)`	A more advanced constructor that lets you turn any object that implements an API interface called `Runnable` into a thread. You see how this constructor is used later in this chapter.
`Thread(Runnable target, String name)`	Creates a thread from any object that implements `Runnable` and assigns the specified name to the thread.

Thread class

`static int activeCount()`	Returns the number of active threads.
`static int enumerate(Thread[] t)`	Fills the specified array with a copy of each active thread. The return value is the number of threads added to the array.
`String getName()`	Returns the name of the thread.
`int getPriority()`	Returns the thread's priority.
`void interrupt()`	Interrupts this thread.
`boolean isInterrupted()`	Checks to see if the thread has been interrupted.
`void setPriority(int priority)`	Sets the thread's priority.
`void setName(String name)`	Sets the thread's name.
`static void Sleep (int milliseconds)`	Causes the currently executing thread to sleep for the specified number of milliseconds.
`void run()`	This method is called when the thread is started. Place the code that you want the thread to execute inside this method.
`void start()`	Starts the thread.
`static void yield()`	Causes the currently executing thread to yield to other threads that are waiting to execute.

Implements Runnable

- Membuat class yang implements Runnable
- Menggunakan method run() dan mengimplementasikan codingnya
- Buat instance dari Thread dan melewatkan object Runnable ke parameter konstruktor Thread(Runnable target) atau dan namanya ke konstruktor Thread(Runnable target,String name)
- Panggil method start()

Contoh

```
class theThread implements Runnable{
     Thread t;

     public theThread(){
          t = new Thread(this,"Demo Thread");
          System.out.println("Thread anak mulai = "+t);
          t.start();
     }

     public void run(){
          try{
               for(int i=5;i>=1;i--){
                    System.out.println("Thread anak : "+i);

                    Thread.sleep(500);
               }
          } catch (InterruptedException ie){
               System.out.println("Thread anak diinterupsi");
          }
          System.out.println("Thread anak selesai.");
     }
}

public class DemoThread{
     public static void main(String[] args){
          new theThread();
          System.out.println("Thread utama mulai");
          try{
               for(int i=0;i<10;i++){
                    System.out.println("Thread utama : "+(i+1));
                    Thread.sleep(1000);
               }
          }catch (InterruptedException ie){
               System.out.println("Thread utama diinterupsi");
          }
          System.out.println("Thread utama selesai.");
     }
}
```

Ouput:

```
C:\>java DemoThread
Thread anak mulai = Thread[Demo Thread,5,main]
Thread utama mulai
Thread utama : 1
Thread anak : 5
Thread anak : 4
Thread utama : 2
Thread anak : 3
Thread anak : 2
Thread utama : 3
Thread anak : 1
Thread anak selesai.
Thread utama : 4
Thread utama : 5
Thread utama : 6
Thread utama : 7
Thread utama : 8
Thread utama : 9
Thread utama : 10
Thread utama selesai.
```

Contoh (2)

```
class myThread implements Runnable{
      private static int count = 0;
      private int number = ++this.count;

      public myThread(){
            System.out.println("Thread dimulai");
            System.out.println("Thread ke "+this.number);
      }

      public void run(){
            for(int i=1;i<=10;i++){
                  System.out.println("T"+this.number+" : "+i);
            }
      }
}

public class DemoThread2{
      public static void main(String[] args){
            for(int i=5;i>=1;i--){
                  Thread t = new Thread(new myThread());
                  t.start();
            }
            System.out.println("Semua Thread telah dibuat");
      }
}
```

Daur hidup Thread (Java)

- Buat Thread baru (new Thread)
- Thread akan dimulai pada saat dibpanggil dengan method start
- Memulai menjalankan Threading (run)
- Method start() akan mengalokasikan resource untuk menjalankan thread, menjadwalkannya, dana memanggil fungsi run(). Setelah distart, maka Thread akan berada dalam status running. Thread dapat berjalan pada satu waktu secara bersamaan, maka Java akan mengatur penjadwalan yang memungkinkan resource digunakan oleh seluruh Thread yang ada. Hal ini yang dapat meningkatkan efisiensi CPU dan performa sebuah aplikasi.
- Membuat Thread yang bersifat blocked
- Thread akan berhenti jika terjadi : blocking I/O, method sleep dipanggil, method wait dipanggil.
- Thread berhenti berjalan (stop)

Subclassing Thread

- Typical setup:
  - Subclass thread, defining a constructor which takes reference to any data structures that the thread may need access to.
  - Constructor sets up members.
  - Override run to do the work of the thread.
- Typical usage:
  - Create instance of the thread subclass.
  - Call start() method of the instance.

Extends Thread

- Kelas java.lang.Thread merupakan kelas yang sudah mengimplemtasikan interface Runnable.
- Perbedaan mendasar kedua metode pembuatanThread adalah bahwa kelas yang mengekstends Thread dapat langsung dijalankan Threadnya : new Thread().start().

Example

```
Thread firstThread = new Thread();
Thread secondThread = new Thread("namedThread");
System.out.println(firstThread.getName());
System.out.println(secondThread.getName());
```

The output from the above lines would be:

```
Thread-0
namedThread
```

Contoh

```
public class ThreadShowName extends Thread
{
    public static void main (String[] args)
    {
        ThreadShowName thread1, thread2;

        thread1 = new ThreadShowName();
        thread2 = new ThreadShowName();

        thread1.start();    //Will call run.
        thread2.start();    //Will call run.
    }

    public void run()
    {
        int pause;
        for (int i=0; i<10; i++)
        {
            try
            {
                System.out.println(
                    getName()+" being executed.");

                pause = (int)(Math.random()*3000);

                sleep(pause);       //0-3 seconds.
            }
            catch (InterruptedException interruptEx)
            {

                System.out.println(interruptEx);
            }
        }
    }
}
```

Contoh (2)

```
class myThread2 extends Thread{
      private static int count = 0;
      private int number = ++this.count;

      public myThread2(){
            System.out.println("Thread dimulai");
            System.out.println("Thread ke "+this.number);
      }

      public void run(){
            for(int i=1;i<=10;i++){
                  System.out.println("T"+this.number+" : "+i);
            }
      }
}

public class DemoThread2{
      public static void main(String[] args){
            for(int i=5;i>=1;i--){
                  new myThread2().start();
            }
            System.out.println("Semua Thread telah dibuat");
      }
}
```

Method Control Thread

- start(); untuk memulai eksekusi Thread.
- stop(); untuk mengakhiri eksekusi Thread.
- suspend(); untuk menghentikan sementara waktu
- resume(); untuk menjalankan kembali Thread yang disuspend.
- sleep(); untuk menghentikan Thread untuk sekian milidetik.

Contoh

```
class myThread3 extends Thread{
	private static int i=0;
	public void run(){
		while(true) i++;
	}
	public int getI() { return i; }
}

public class DemoThread3{
	public static void main(String[] args){
		boolean isSuspend = false;
		myThread3 mt3 = new myThread3();
		mt3.start();
		for(int j=0;j<100;j++){
			if (j%25 == 0){
				if(isSuspend){
			System.out.println("Sebelum resume j = "+j+", i = "+mt3.getI());
			mt3.resume();
			System.out.println("Setelah resume j = "+j+", i = "+mt3.getI());
				} else {
			System.out.println("Sebelum suspend j = "+j+", i = "+mt3.getI());
			mt3.suspend();
			System.out.println("Setelah suspend j = "+j+", i = "+mt3.getI());
				}
			isSuspend = !isSuspend;
			}
		}
		mt3.stop();
	}
}
```

Thread Synchronization

- An important consideration when designing multi-threaded applications is conflict over access to data.
- If two threads are fighting for the same resource, and a mechanism to resolve access conflicts is not put into place, the integrity of the application is at stake.
- If a class has at least one synchronized methods, each instance of it has a monitor. A monitor is an object that can block threads and notify them when it is available.
- Built into the Java language are two mechanisms for preventing concurrent access to resources:
  - method-level synchronization
  - block-level synchronization.

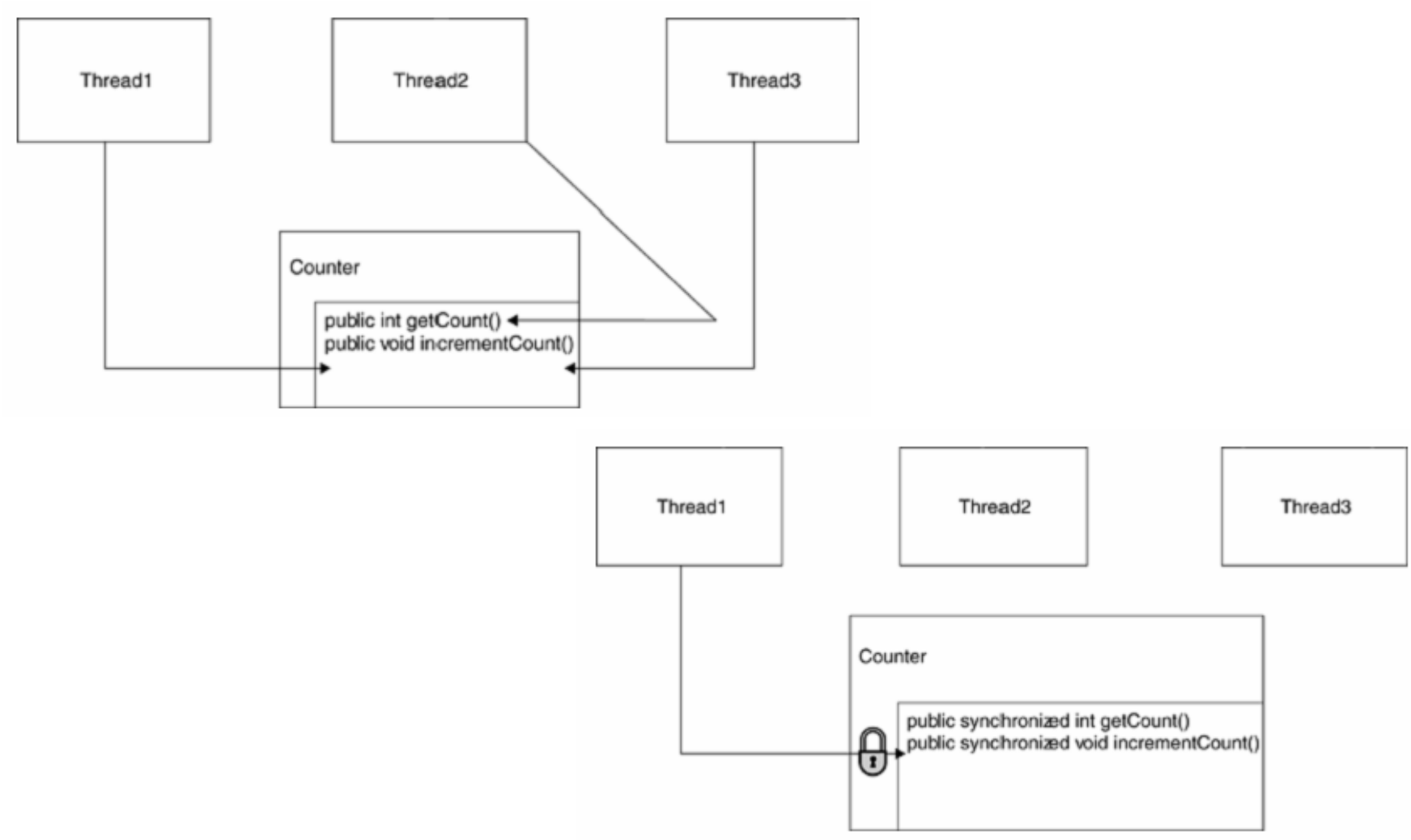
Method-level synchronization

- prevents two threads from executing methods on an object at the same time.

```
public class SomeClass
{
      public synchronized void changeData( ... )
      {
            .........
      }

      public synchronized Object getData ( ... )
      {
          ............
      }
}
```

Not Safe vs Safe Thread



Method Level synchronization

- is an effective means of preventing concurrent access to resources.
- But what if the resource has not been designed as thread-safe, and is a preexisting class that the developer cannot modify (such as a class in the Java API, or a third-party library)?
- Block-level synchronization, in this case, is the best option.

```
synchronized (Object o)
{
        ......
}
```

Contoh

Thread Priority

- On a single CPU threads actually run one at a time in such a way as to provide an illusion of concurrency.
- Execution of multiple threads on a single CPU, in some order, is called scheduling.
- The Java runtime supports a very simple scheduling algorithm (fixed priority scheduling). This algorithm schedules threads based on their priority relative to other runnable threads.

Thread Priority

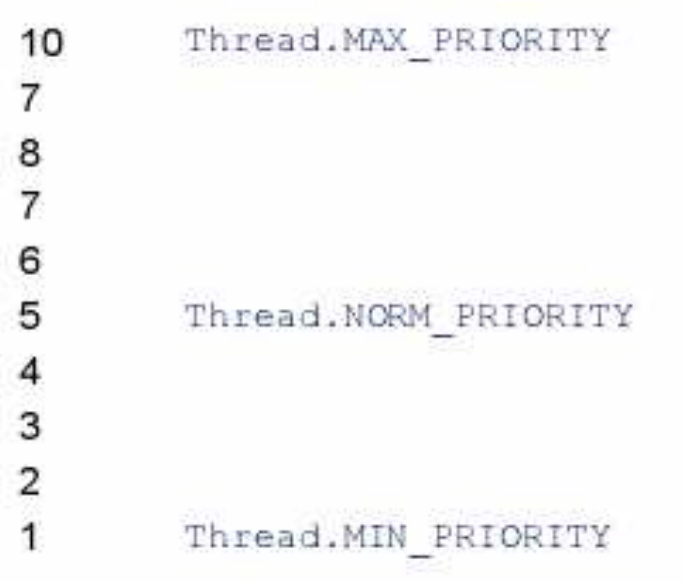
- The runtime system chooses the runnable thread with the highest priority for execution
- If two threads of the same priority are waiting for the CPU, the scheduler chooses one of them to run in a round-robin fashion.

- The chosen thread will run until:
  - A higher priority thread becomes runnable.
  - It yields, or its runmethod exits.
  - On systems that support time-slicing, its time allotment has expired

Thread Priority

- When a Java thread is created, it inherits its priority from the thread that created it.
- You can modify a thread's priority at any time after its creation using the setPriority method.
- Thread priorities are integers ranging between MIN_PRIORITY and MAX_PRIORITY (constants defined in the Threadclass). The higher the integer, the higher the priority.
- Use priority only to affect scheduling policy for efficiency purposes.
- Do not rely on thread priority for algorithm correctness.

Thread Priority



- **Assigning a Thread Priority**

```
Thread t = new Thread (runnable);
t.setPriority ( Thread.MIN_PRIORITY );
t.start();
```

- **Obtaining the Current Thread Priority**

```
Thread t = Thread.currentThread();
System.out.println ("Priority : " +
t.getPriority());
```

- **Limiting Thread Priority**

```
ThreadGroup group = new ThreadGroup (
"mygroup" );
group.setMaximumPriority(8);
```