Multithreading

Thread Control Mechanism

Agenda

1 Thread Control Mechanism

Objectives

At the end of this module, you will be able to:

- Thread Control Mechanism

Thread Control Mechanism

Control Thread Execution

- Two ways exist by which you can determine whether a thread has finished:

- The **isAlive( )** method will return true if the thread upon which it is called is still running; else it will return false

- The **join( )** method waits until the thread on which it is called terminates.
- Syntax:
- final boolean isAlive()
- final void join() throws InterruptedException

Control Thread Execution (Contd.).

```
public class DemoThread implements Runnable {
String name;
Thread thread;
DemoThread(String threadname) {
name = threadname;
thread = new Thread(this, name);
System.out.println("New Thread: " + thread);
thread.start();
}
```

Control Thread Execution (Contd.).

```
public void run() {
try {
      for(int i=5; i>0; i--) {
System.out.println("Child Thread: " + i);
Thread.sleep(1000); }
     }
catch (InterruptedException e) {
        System.out.println(name + "Interrupted");
}
System.out.println(name +"Exiting");
}}
```

Control Thread Execution (Contd.).

```
public class MultiThreadImpl {
public static void main(String args[]) {
DemoThread t1 = new DemoThread("One");
DemoThread t2 = new DemoThread("Two");
DemoThread t3 = new DemoThread("Three");
System.out.println("Thread One is alive: " +
t1.thread.isAlive());
System.out.println("Thread Two is alive: " +
t2.thread.isAlive());
System.out.println("Thread Three is alive: "  +
t1.thread.isAlive());
```

Control Thread Execution (Contd.).

```
       try {
      System.out.println("Waiting for child  threads to
finish");
      t1.thread.join();
      t2.thread.join();
      t3.thread.join();
}
catch (InterruptedException e) {
      System.out.println("Main thread interrupted");
}
```

Control Thread Execution (Contd.).

```
System.out.println("Thread One is alive: " +
t1.thread.isAlive());
System.out.println("Thread One is alive: " +
t1.thread.isAlive());
System.out.println("Thread One is alive: " +
t1.thread.isAlive());
System.out.println("Main thread exiting");
}
}
```

Control Thread Execution (Contd.).

Output:

```
New Thread: Thread[One,5,main]
New Thread: Thread[Two,5,main]
Child Thread: 5
New Thread: Thread[Three,5,main]
Thread One is alive: true
Thread Two is alive: true
Thread Three is alive: true
Waiting for child  threads to finish
Child Thread: 5
Child Thread: 5
Child Thread: 4
Child Thread: 4
Child Thread: 4
```

Continued….

```
Child Thread: 3
Child Thread: 3
Child Thread: 3
Child Thread: 2
Child Thread: 2
Child Thread: 2
Child Thread: 1
Child Thread: 1
Child Thread: 1
ThreeExiting
OneExiting
TwoExiting
Thread One is alive: false
Thread One is alive: false
Thread One is alive: false
Main thread exiting
```

Assignment

Summary

- Different Thread control mechanisms

Thank You

Sensitivity: Internal & Restricted

Multithreading

Thread Priorities

Agenda

1 Thread Priorities

Objectives

At the end of this module, you will be able to:

- Thread Priorities

Thread Priorities

A Thought:

- When there are multiple threads running at the same time, how the CPU decides which thread should be given more time to execute and complete first?

Thread Priorities

- A thread priority decides:
  - The importance of a particular thread, as compared to the other threads
  - When to switch from one running thread to another

- The term that is used for switching from one thread to another is **context switch**

- Threads which have higher priority are usually executed in preference to threads that have lower priority

Thread Priorities (Contd.).

- When the thread scheduler has to pick up from several threads that are runnable, it will check the thread priority and will decide when a particular thread has to run
- The threads that have higher-priority usually get more CPU time as compared to lower-priority threads
- A higher priority thread can also preempt a lower priority thread
- Actually, threads of equal priority should evenly split the CPU time

Thread Priorities (Contd.).

- Every thread has a priority
- When a thread is created it inherits the priority of the thread that created it
- The methods for accessing and setting priority are as follows:

```
public final int getPriority( );
public final void setPriority (int level);
```

Thread Priorities (Contd.).

- JVM selects a Runnable thread with the highest priority to run
- All Java threads have a priority in the range 1-10
- Normal priority i.e.. priority by default is 5
- Top priority is 10, lowest priority is 1
- Thread.MIN_PRIORITY - minimum thread priority
- Thread.MAX_PRIORITY - maximum thread priority
- Thread.NORM_PRIORITY - normal thread priority

Thread Priorities (Contd.).

- When a new Java thread is created it has the same priority as the thread which created it
- Thread priority can be changed by the **setPriority()** method

```
thread1.setPriority(Thread.NORM_PRIORITY + 1);
thread2.setPriority(Thread.NORM_PRIORITY -1);
thread3.setPriority(Thread.MAX_PRIORITY - 1);

thread1.start();
thread2.start();
thread3.start();
```

Example on Thread Priority

To demonstrate thread priority, we will use the same demo from the ‘creating multiple threads’ topic.

Modifying the program to include thread priority:

```
public class ThreadDemo implements Runnable {
public void run() {
for(int counter=1;counter<=100;counter++){
System.out.println(Thread.currentThread().getName()+"thread is running..."+counter);
}}
public static void main(String args[]) {
ThreadDemo threadDemo = new ThreadDemo();
Thread t1 = new Thread(threadDemo,"First");
t1.setPriority(Thread.MAX_PRIORITY);
Thread t2 = new Thread(threadDemo,"Second");
t2.setPriority(Thread.MIN_PRIORITY);
t1.start();
t2.start();
}}
```

Setting MAX priority to t1

Setting MIN priority to t2

Example on Thread Priority

Output:

```
Firstthread is running...1
Firstthread is running...2
Secondthread is running...1
Firstthread is running...3
Firstthread is running...4
Firstthread is running...5
Firstthread is running...6
Firstthread is running...7
Firstthread is running...8
Firstthread is running...9
Firstthread is running...10
Firstthread is running...11
Firstthread is running...12
Firstthread is running...13
Secondthread is running...2
Firstthread is running...14
Firstthread is running...15
Firstthread is running...16
Firstthread is running...17
Firstthread is running...18
Firstthread is running...19
Firstthread is running...20
```

We can see that the First thread is given more execution time than the Second thread since First thread is having MAX priority.

Deciding on a Context Switch

- A thread can voluntarily relinquish control by explicitly yielding, sleeping, or blocking on pending Input/ Output
- All threads are examined and the highest-priority thread that is ready to run is given by the CPU
- A thread can be preempted by a higher priority thread
- A lower-priority thread that does not yield the processor is superseded, or preempted by a higher-priority thread

This is called preemptive multitasking.

- **When two threads with the same priority are competing for CPU time, threads are time-sliced in round-robin fashion in case of Windows like OSs**

Assignment

Summary

- Different Thread priorities
- Setting the priority of the thread
- Getting thread priority