VIT-AP UNIVERSITY

Course code : **CSE2005**

Course title : **Object Oriented Programming**

Thread Control Mechanism

Objectives

This session will give the knowledge about

- Thread Control Mechanism
- Thread Priorities

Control Thread Methods

Core Java provides complete control over multithreaded program. You can develop a multithreaded program which can be suspended, resumed, or stopped completely based on your requirements.

public void suspend()

This method puts a thread in the suspended state and can be resumed using resume() method.

public void stop()

This method stops a thread completely.

Control Thread Methods

public void resume()

This method resumes a thread, which was suspended using suspend() method.

public void wait()

Causes the current thread to wait until another thread invokes the notify().

public void notify()

Wakes up a single thread that is waiting on this object's monitor.

Control Thread Methods

```
class MyThread extends Thread {
        public void run() {
                try {
                        for (int i = 0; i < 5; i++) {
                        System.out.println(Thread.currentThread().getName() + i);
                                Thread.sleep(1000);
                        }
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
```

Control Thread Methods

```
public class ThreadDemo {
        public static void main(String args[]) {
                MyThread t1 = new MyThread();          t1.setName("first");
                MyThread t2 = new MyThread();          t2.setName("second");
                MyThread t3 = new MyThread();          t3.setName("third");
                t1.start();       // call run() method
                t2.start();
                t2.suspend(); // suspend t2 thread
                t3.start();       // call run() method
                t2.resume();   // resume t2 thread
        }
}
```

Control Thread Methods

```
public class ThreadDemo {
        public static void main(String args[]) {
                MyThread t1 = new MyThread();          t1.setName("first");
                MyThread t2 = new MyThread();          t2.setName("second");
                MyThread t3 = new MyThread();          t3.setName("third");
                t1.start();       // call run() method
                t2.start();
                t2.stop(); // stop t2 thread
                t3.start();       // call run() method
                t2.resume();   // useless not possible to resume
        }
}
```

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

```
class MyThread extends Thread {
        public void run() {
                try {
                        for (int i = 0; i < 5; i++) {
                System.out.println(Thread.currentThread().getName()+i);
                                Thread.sleep(1000);
                        }
                } catch (InterruptedException e) {
                        e.printStackTrace();
                }
        }
}
```

Control Thread Execution

```
public class ThreadDemo {
        public static void main(String args[]) {
                MyThread t1 = new MyThread();
                t1.setName("first");
                t1.start();
                MyThread t2 = new MyThread();
                t2.setName("second");
                t2.start();
                MyThread t3 = new MyThread();
                t3.setName("third");
                t3.start();
```

Control Thread Execution

```
System.out.println("first:"+t1.isAlive());
System.out.println("second:"+t2.isAlive());
System.out.println("third:"+t2.isAlive());
try{
        t1.join();
        t2.join();
        t3.join();
}catch (InterruptedException e) {
        e.printStackTrace();
}
System.out.println("first:"+t1.isAlive());
```

Control Thread Execution

```
                System.out.println("second:"+t2.isAlive());
                System.out.println("third:"+t2.isAlive());
        }
}
```

first0

third0

second0

first:true

second:true

Control Thread Execution

third:true
second1
first1
third1
second2
third2
first2
first3
third3
second3
first4
third4
second4
first:false
second:false
third:false

VIT-AP UNIVERSITY

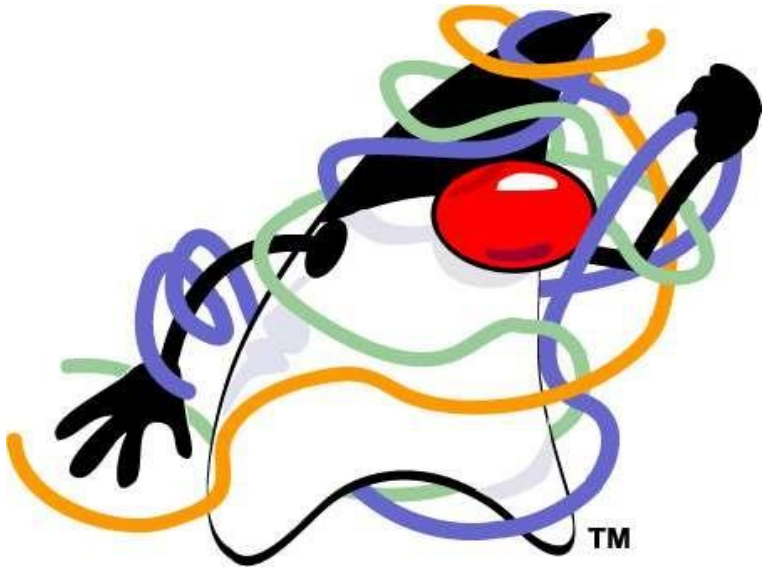
Course code : **CSE2005**

Course title : **Object Oriented Programming**

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

- 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 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

- 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

- 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

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

Thread Priorities

```
public class ThreadDemo {
        public static void main(String args[]) {
                MyThread t1 = new MyThread();           t1.setName("first");
                MyThread t2 = new MyThread();           t2.setName("second");
                MyThread t3 = new MyThread();           t3.setName("third");
                t3.setPriority(Thread.MAX_PRIORITY);
                t2.setPriority(Thread.MIN_PRIORITY);
                t1.start();
                t2.start();
                t3.start();
        }
}
```

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 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

Summary

We have discussed about

- Thread Control Mechanism
- Thread Priorities