



**CSE 2202 : Object Oriented
Programming
(OOP)**

Introduction to Thread

2018



Computer Science & Engineering Program
The School of EE & Computing
Adama Science & Technology University

- Threads are sequences of code that can be executed independently alongside one another.
- The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. Thus a thread is an independent sequential flow of control within a process.

- Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU.
- Multitasking can be achieved by two ways:
 - Process-based
Multitasking(Multiprocessing)
 - Thread-based
Multitasking(Multithreading)

1) **Process-based Multitasking (Multiprocessing)**

- Each process have its own address in memory i.e. each process allocates separate memory area.
- Process is heavyweight.
- Cost of communication between the process is high.
- Switching from one process to another require some time for saving and loading registers, memory maps, updating lists etc.

2) **Thread-based Multitasking (Multithreading)**

- Threads share the same address space.
- Thread is lightweight.
- Cost of communication between the thread is low.



Multithreading in java

- **Multithreading in java** is a process of executing multiple threads simultaneously.
- Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.
- But we use multithreading than multiprocessing because threads share a common memory area.
- They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process

- 1) It **doesn't block the user** because threads are independent and you can perform multiple operations at same time.
- 2) You **can perform many operations together so it saves time.**
- 3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

- A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.
- But for better understanding the threads, we are explaining it in the 5 states.
- The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:
 - I. New
 - II. Runnable
 - III. Non-Runnable (Blocked)
 - IV. Terminated

- 1) **New**: the thread object has been created but it has not been started yet so it cannot run.
- 2) **Runnable**: this means that a thread can be run when the time-slicing mechanism has CPU cycles available for the thread. Thus, the thread might or might not be running, but there is nothing to prevent it from being run if the scheduler can arrange it. It is not dead or blocked.
- 3) **Terminated/Dead**: the normal way for a thread to die is by returning from its run() method or when its run() method exits.
- 4) **Blocked**: the thread could be run but there is something that prevents it. While a thread is in the blocked state the scheduler will simply skip over it and not give any CPU time.

There are two ways to create a thread:

- By extending Thread class
- By implementing Runnable interface.

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()`.

public void run(): is used to perform action for a thread.

Java Thread Example by extending Thread class



```
class Multi extends Thread{  
public void run(){  
for(int i=1;i<5;i++){  
    System.out.println(i);  
}  
}  
public static void main(String args[]){  
Multi t1=new Multi();  
t1.start();  
}  
}
```

Output:
1234

Starting a thread:

- **start() method** of Thread class is used to start a newly created thread. It performs following tasks:
- A new thread starts(with new callstack).
- The thread moves from New state to the Runnable state.
- When the thread gets a chance to execute, its target run() method will run

Java Thread Example by implementing Runnable interface

```
class Multi3 implements Runnable{  
public void run(){  
    for(int i=1;i<5;i++){  
        System.out.print(i);  
    }  
}  
  
public static void main(String args[]){  
    Multi3 m1=new Multi3();  
    Thread t1 =new Thread(m1);  
    t1.start();  
}  
}
```

Output:
1234

- **Thread scheduler** in java is the part of the JVM that decides which thread should run.
- There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
- Only one thread at a time can run in a single process.
- The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

- **Under preemptive scheduling**, the highest priority task executes until it enters the waiting or dead states or a higher priority task comes into existence.
- **Under time slicing**, a task executes for a predefined slice of time and then reenters the pool of ready tasks. The scheduler then determines which task should execute next, based on priority and other factors.



Example of sleep method in java

The sleep() method of Thread class is used to sleep a thread for the specified amount of time.

```
class TestSleepMethod1 extends Thread{  
    public void run(){  
        for(int i=1;i<5;i++){  
            try{Thread.sleep(500);} catch(InterruptedException e)  
{System.out.println(e);}  
            System.out.print(i);  
        }  
    }  
    public static void main(String args[]){  
        TestSleepMethod1 t1=new TestSleepMethod1();  
        TestSleepMethod1 t2=new TestSleepMethod1();  
  
        t1.start();  
        t2.start();  
    }  
}
```

Output:
1 1 2 2 3 3 4
4C

Call Run method Using start vs Using object ASTU

```
class TestCallRun2 extends Thread{  
public void run(){  
    for(int i=1;i<5;i++){  
        try{Thread.sleep(500);} catch(InterruptedException e)  
{System.out.println(e);}  
        System.out.println(i);  
    }  
}  
  
public static void main(String args[]){  
    TestCallRun2 t1=new TestCallRun2();  
    TestCallRun2 t2=new TestCallRun2();  
  
    t1.run();  
    t2.run();  
}
```

Output :
12341234



Join method in a thread class

```
class TestJoinMethod1 extends Thread{
    public void run(){
        for(int i=1;i<=5;i++){
            try{
                Thread.sleep(500);
            }catch(Exception e){System.out.println(e);}
            System.out.print(i);
        }
    }
    public static void main(String args[]){
        TestJoinMethod1 t1=new TestJoinMethod1();
        TestJoinMethod1 t2=new TestJoinMethod1();
        TestJoinMethod1 t3=new TestJoinMethod1();
        t1.start();
        try{
            t1.join();
        }catch(Exception e){System.out.println(e);}

        t2.start();
        t3.start();
    }
}
```

Output:1 2 3 4 5 1 1 2 2 3 3 4 4 5 5

Change Thread name

```
class TestJoinMethod3 extends Thread{
    public void run(){
        System.out.println("running...");
    }
    public static void main(String args[]){
        TestJoinMethod3 t1=new TestJoinMethod3();
        TestJoinMethod3 t2=new TestJoinMethod3();
        System.out.println("Name of t1:"+t1.getName());
        System.out.println("Name of t2:"+t2.getName());
        System.out.println("id of t1:"+t1.getId());

        t1.start();
        t2.start();

        t1.setName("CSE Thread");
        System.out.println("After changing name of t1:"+t1.getName());
    }
}
```



Multiple task with multiple Thread

```
class Simple1 extends Thread{  
    public void run(){  
        System.out.println("task one");  
    }  
}
```

```
class Simple2 extends Thread{  
    public void run(){  
        System.out.println("task two");  
    }  
}
```

```
class TestMultitasking3{  
    public static void main(String args[]){  
        Simple1 t1=new Simple1();  
        Simple2 t2=new Simple2();  
  
        t1.start();  
        t2.start();  
    }  
}
```

Output:
Task one
task two

- Synchronization in java is the *capability to control the access of multiple threads to any shared resource.*
 - Java Synchronization is better option where we want to allow only one thread to access the shared resource.
- The synchronization is mainly used to
- To prevent thread interference.
 - To prevent consistency problem.

- There are two types of thread synchronization mutual exclusive and inter-thread communication.
- Mutual Exclusive
 - Synchronized method.
 - Synchronized block.
 - static synchronization.
- Cooperation (Inter-thread communication in java)



with out synchronized method

```
class Table{

    void printTable(int n){//method not
synchronized
    for(int i=1;i<=5;i++){
        System.out.println(n*i);
        try{
            Thread.sleep(400);
        }catch(Exception e)
    {System.out.println(e);}
    }
}

class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(5);
    }
}
```

```
class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
        this.t=t; }
    public void run(){
        t.printTable(100);
    } }

    class TestSynchronization1{
        public static void main(String args[]){
            Table obj = new Table();//only one object
            MyThread1 t1=new MyThread1(obj);
            MyThread2 t2=new MyThread2(obj);
            t1.start();
            t2.start();
        } }
```

Output:

5 100 10 200 15 300 20 400 25
500



with synchronized method

```
class Table{

synchronized void printTable(int
n){//method not synchronized
    for(int i=1;i<=5;i++){
        System.out.println(n*i);
        try{
            Thread.sleep(400);
        }catch(Exception e)
    {System.out.println(e);}
    }
}

class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(5);
    }
}
```

```
class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
        this.t=t;  }
    public void run(){
        t.printTable(100);
    }  }

    class TestSynchronization1{
        public static void main(String args[]){
            Table obj = new Table();//only one object
            MyThread1 t1=new MyThread1(obj);
            MyThread2 t2=new MyThread2(obj);
            t1.start();
            t2.start();
        }  }
```

Output:

5 10 15 20 25 100 200 300 400
500

- **Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.
- Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.
- It is implemented by following methods of **Object class**:
 - wait(): Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
 - notify(): Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened.
 - notifyAll(): Wakes up all threads that are waiting on this object's monitor.

Understanding the process of inter-thread communication



The point to point explanation of the above diagram is as follows:

- Threads enter to acquire lock.
- Lock is acquired by one thread.
- Now thread goes to waiting state if you call wait() method on the object. Otherwise it releases the lock and exits.
- If you call notify() or notifyAll() method, thread moves to the notified state (runnable state).
- Now thread is available to acquire lock.
- After completion of the task, thread releases the lock and exits the monitor state of the object.

Inter-Thread Communication example



ASTU

```
class Customer{  
int amount=10000;  
  
synchronized void withdraw(int amount){  
    System.out.println("going to withdraw...");  
  
    if(this.amount<amount){  
        System.out.println("Less balance; waiting for deposit...");  
        try{wait();}catch(Exception e){}  
    }  
    this.amount-=amount;  
    System.out.println("withdraw completed...");  
}  
  
synchronized void deposit(int amount){  
    System.out.println("going to deposit...");  
    this.amount+=amount;  
    System.out.println("deposit completed... ");  
    notify();  
}  
}
```

```
class Test{  
    public static void main(String  
    args[]){  
        final Customer c=new Custome  
        r();  
        new Thread(){  
            public void run()  
            {c.withdraw(15000);} }.start();  
        new Thread(){  
            public void run()  
            {c.deposit(10000);} }.start();  
    }  
}
```

Output:

```
going to withdraw...  
Less balance; waiting for deposit...  
going to deposit...  
deposit completed...  
withdraw completed...
```

Thank you staying with me!!!

Q?

End of course !!!

Good time with CSE
program!!

If you have any
comment/suggestion?
come

you well