Welcome you all

JAVA PROGRAMMING

DAY : 8

D.Sakthivel
Assistant Professor & Trainer,
KGiSL Micro College
KGiSL Campus, Coimbatore – 641 035.

Day- 8

THREAD

- What is Thread?
- Thread Model
- Thread Methods
- How to Create Thread in Java?
- Thread Priority
- Multithreading

What is Thread?

A **Thread** is a very light-weighted process, or we can say the smallest part of the process that allows a program to operate more efficiently by running multiple tasks simultaneously.

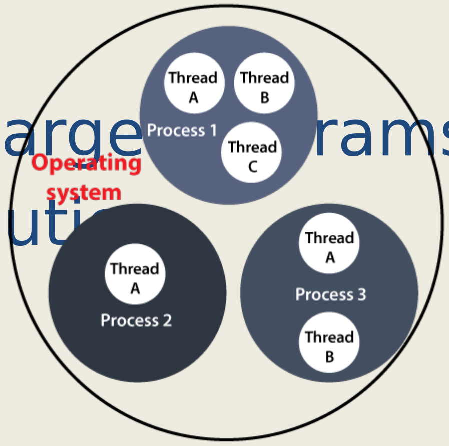
In order to perform complicated tasks in the background, we used the **Thread concept in Java**.

All the tasks are executed without affecting the main program.

In a program or process, all the threads have their own separate path for execution, so each thread of a process is independent.

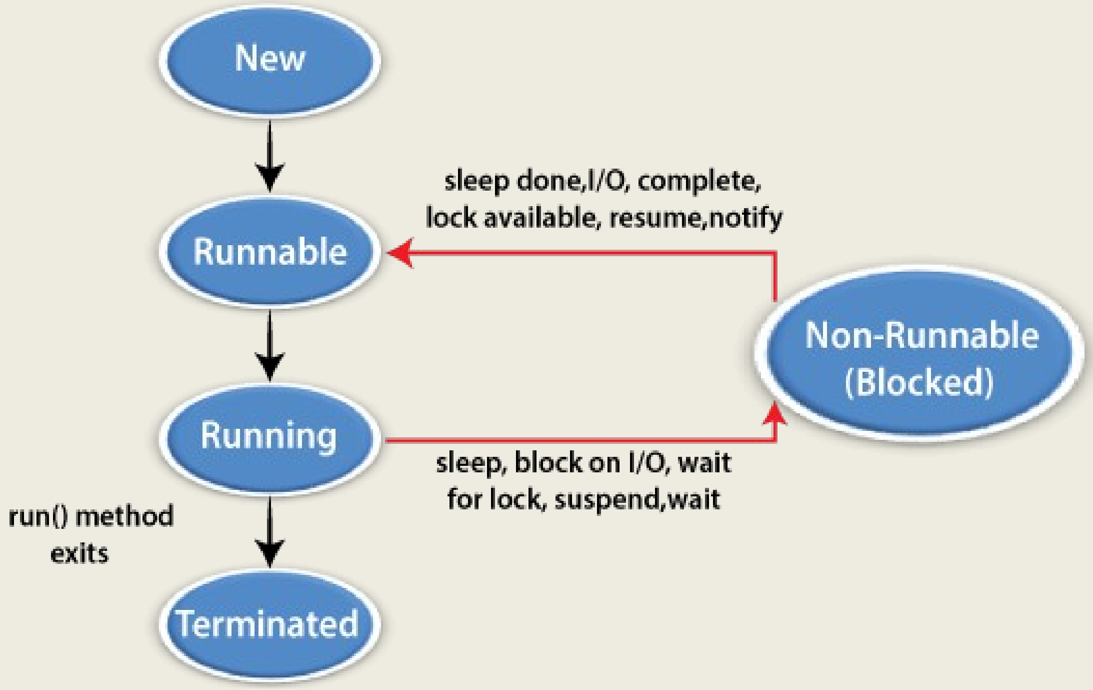
Thread Benefits

- All the threads share a common memory and have their own stack, local variables and program counter.
- When multiple threads are executed in parallel at the same time, this process is known as **Multithreading**.
- It allows to perform more than one task at a time.
- Resource sharing between threads is easy.
- It maximizes CPU utilization
- It reduces the complexity of large programs.
- It increases the speed of execution.



Thread Model

Just like a process, a thread exists in several states. These states are as follows:


New
Runnable
Running
Terminated
Non-Runnable (Blocked)
sleep done,I/O, complete, lock available, resume,notify
sleep, block on I/O, wait for lock, suspend,wait
run() method exits

Thread Model

1) New (Ready to run)

A thread is in **New** when it gets CPU time.

2) Running

A thread is in **a Running** state when it is under execution.

3) Suspended

A thread is in the **Suspended** state when it is temporarily inactive or under execution.

4) Blocked

A thread is in the **Blocked** state when it is waiting for resources.

5) Terminated

A thread comes in this state when at any given time, it halts its execution immediately.

Creating Thread

There are two methods to create threads. They are

i) **Creating threads by extending Thread class.**

ii) **Creating threads by implementing Runnable interface**

Thread methods

The important thread class methods are

a) **run()**

This method should be overridden in our thread extended the superclass thread. This method contains the statements for the particular thread.

```
public void run(){

statements

}
```

b) start()

This method is used to start the run() method. If the method is already started it throws illegal Thread state exception. The general form is

```
      void start()
```

Thread methods

c) sleep()

This method is used to block the currently executing thread. The general form is.

```
static void sleep(longint a)
```

Where,

a- integer value. This gives the

d) interrupt()

This method is used to interrupt the currently running thread. The general form is time for sleeping in milli seconds

```
      void interrupt()
```

Thread methods

e) interrupted()

The general form is

```
static boolean interrupted()
```

This method returns true, if the current thread is interrupted else false.

f) isAlive()

This method is used to check whether the thread is running or not. The general form is

```
Boolean isActive()
```

This method returns true, if the thread is running else false.

g) stop()

This method is used to stop the running thread. The

```
void stop()
```

Thread methods

h) yield()

This method is used to bring the stopped thread to run mode. The general form is

```
void yield()
```

i) wait()

This method is used to stop the currently running thread. The general form is

```
void wait()
```

Creating threads by extending Thread class

The steps given below are used to create a thread by extending thread class.

- Define a **thread subclass** by **extending** from the s**uper class thread.**
- Override the **thread class method run()** in the extended class with the statements to be executed by the thread.
- Write the main class and define thread objects.
- Using the created thread objects start the thread using start() method.

Sample code to create Threads by Extending Thread Class:

```
import java.io.*;
import java.util.*;

public class ABC extends Thread
{
     // initiated run method for Thread
    public void run()
    {
        System.out.println("Thread Started Running...");
    }
    public static void main(String[] args)
    {
        ABC g1 = new ABC();
          // invoking Thread
        g1.run();
    }
}
```

Output

```
Thread Started
Running...
```

Creating threads by implementing runnable interface

The steps given below are used to create a thread by implementing runnable interface

- Define a thread subclass by **implementing from the interface Runnable**
- **Declare the thread class method run()** in the implemented class with the statements to be executed by the thread.
- Write the main class and **define thread objects.**
- Use the **created thread object as the argument** to the **constructor Thread** and start the thread.

Sample code to create Thread by using Runnable Interface:

```
import java.io.*;
import java.util.*;
 public class ABC  implements Runnable
 {
     // method to start Thread
    public void run()
    {
        System.out.println(
            "Thread is Running Successfully");
    }
    public static void main(String[] args)
    {
        ABC g1 = new ABC ();
        // initializing Thread Object
        Thread t1 = new Thread(g1);
        t1.run();
    }
}
```

Output:

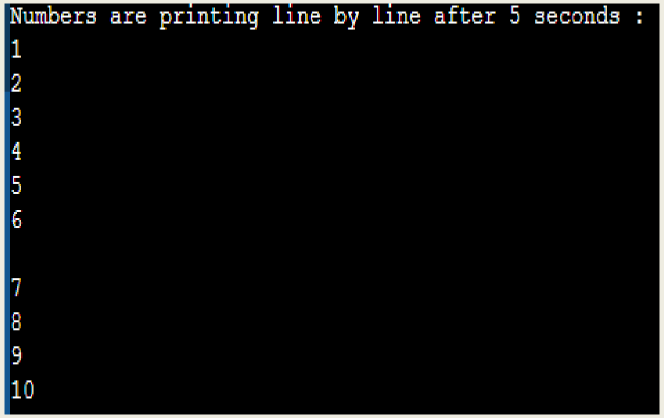
```
Thread is Running
Successfully
```

Interrupting Methods

- An interrupt is an indication to a thread that it should stop what it is doing and do something else.
- How does a thread support its own interruption?
- This depends on what it's currently doing.
- If the thread is frequently invoking methods that throw Interrupted Exception, it simply returns from the run method after it catches that exception.

Interrupting Methods

```
public class Threads
{
public static void main(String[] args)
{
Thread th = new Thread();
System.out.println("Numbers are printing line by line after 5
seconds : ");
try
{
    for(int i = 1;i <= 10;i++)
    {
    System.out.println(i);
    th.sleep(5000);
    }
}
catch(InterruptedException e)
{
    System.out.println("Thread interrupted!");
    e.printStackTrace();
}
}}
```



```
Numbers are printing line by line after 5 seconds :
1
2
3
4
5
6

7
8
9
10
```

Running state:

If a thread is in execution then this state is called running state. This state continues until any one of the following happens after the completion of the execution

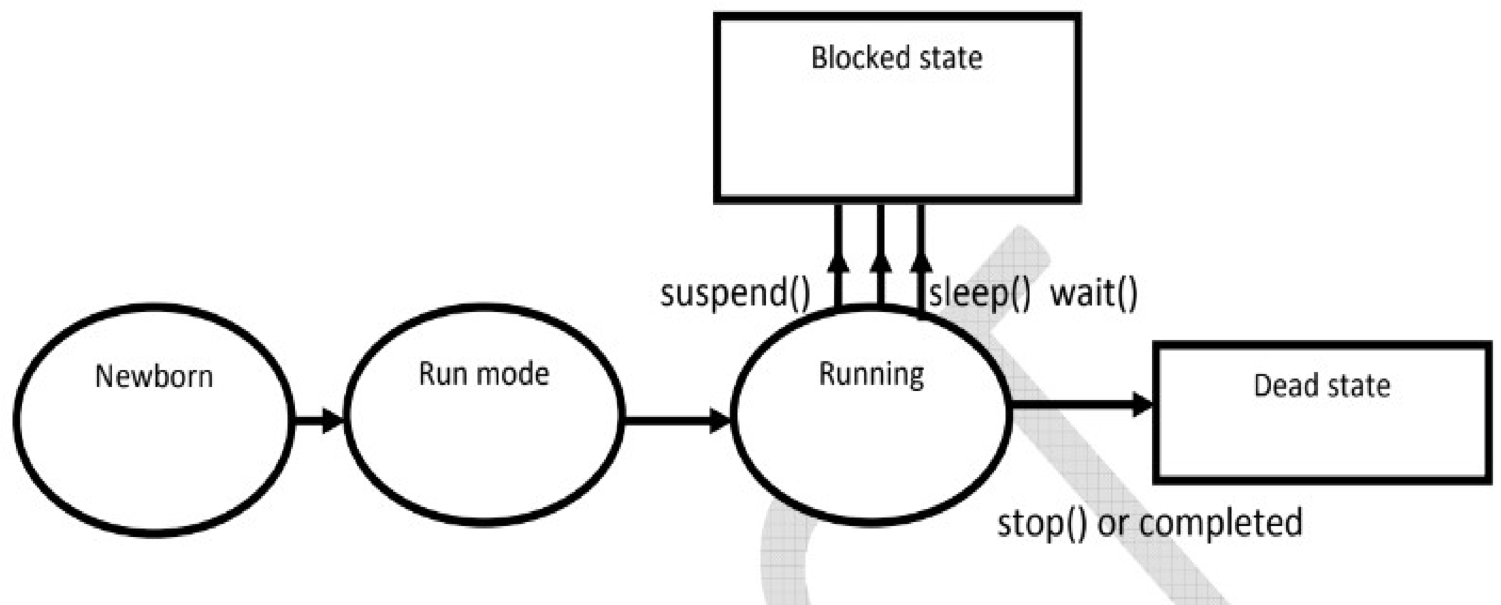
a) when yield() is called

b) when sleep() is called

c) when wait() is called

d) when suspend() is called

The graph given below shows the flow of running state of the thread.



Blocked state:

A thread becomes blocked state if any one of the following method is called while thread is

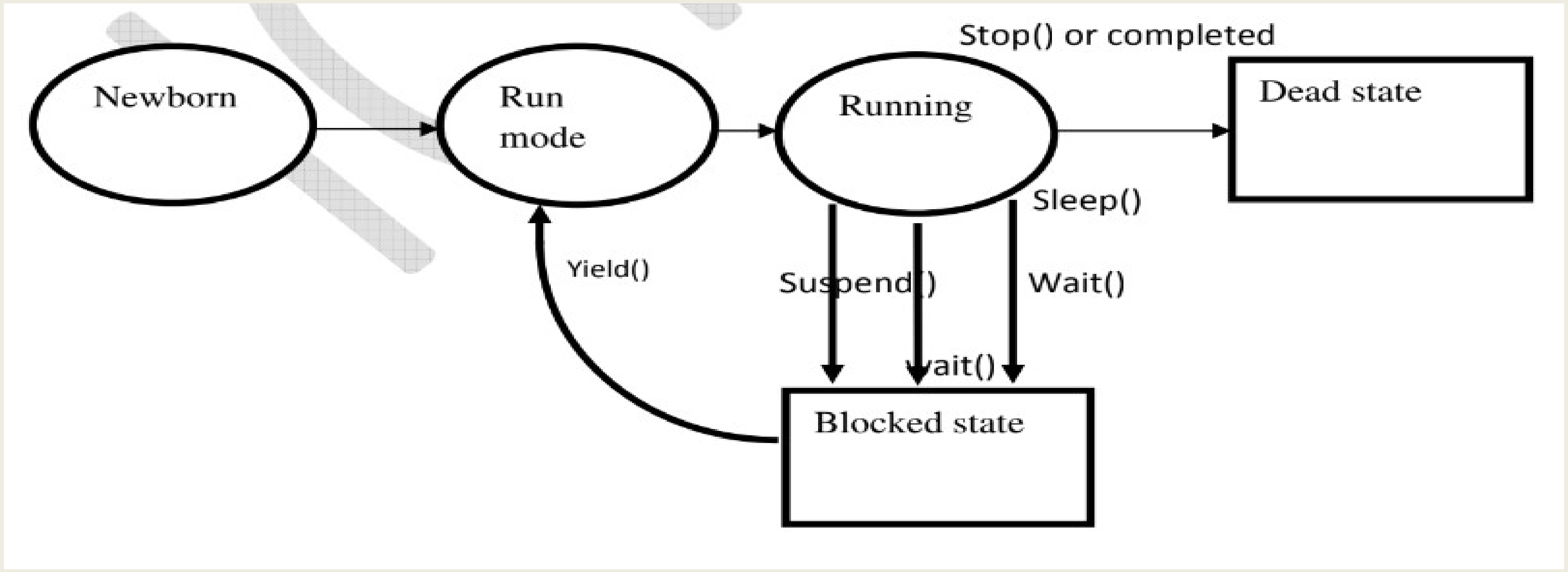
running.

a) sleep()

b) wait()

c) suspend()

From blocked state it comes to run mode state if the yield () method is called. The graph given below shows the flow of the blocked state of the thread.

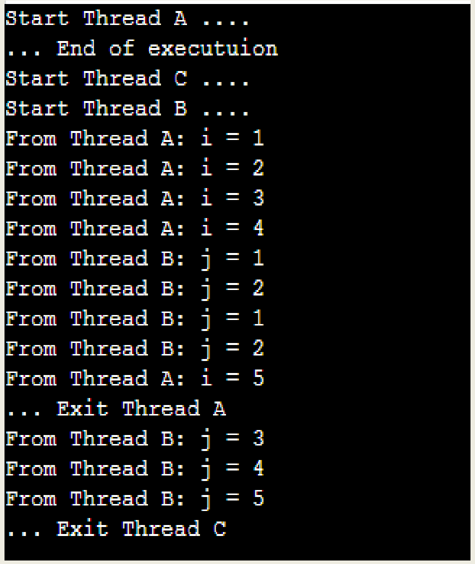


```
/* Use of yield(), stop() and sleep() methods */
class ClassA extends Thread{
    public void run() {
        System.out.println("Start Thread A ....");
        for(int i = 1; i <= 5; i++) {
            if (i==1) yield();
            System.out.println("From Thread A: i = "+ i);
        }
        System.out.println("... Exit Thread A");
    }
}

class ClassB extends Thread{
    public void run() {
        System.out.println("Start Thread B ....");
        for(int j = 1; j <= 5; j++) {
            System.out.println("From Thread B: j = "+ j);
            if (j==2) stop();
        }
        System.out.println("... Exit Thread B");
    }
}
```

```
/* Use of yield(), stop() and sleep() methods */
class ClassC extends Thread{
    public void run() {
        System.out.println("Start Thread C ....");
        for(int k = 1; k <= 5; k++) {
            System.out.println("From Thread B: j = "+ k);
            if (k==3){
                try{
                    sleep(1000);
                }catch(Exception e){}
            }
        }
        System.out.println("... Exit Thread C");
    }
}

public class Thread_State{
    public static void main (String args[]) {
       ClassA t1 = new ClassA();
        ClassB t2 = new ClassB();
        ClassC t3 = new ClassC();
        t1.start(); t2.start(); t3.start();
        System.out.println("... End of executuion ");
    }
}
```



```
Start Thread A ....
... End of executuion
Start Thread C ....
Start Thread B ....
From Thread A: i = 1
From Thread A: i = 2
From Thread A: i = 3
From Thread A: i = 4
From Thread B: j = 1
From Thread B: j = 2
From Thread B: j = 1
From Thread B: j = 2
From Thread A: i = 5
... Exit Thread A
From Thread B: j = 3
From Thread B: j = 4
From Thread B: j = 5
... Exit Thread C
```

Priority of a Thread (Thread Priority)

- Each thread has a priority.
- Priorities are represented by a number between 1 and 10.
- In most cases, the thread scheduler schedules the threads according to their priority (known as preemptive scheduling).
- But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.
- Note that not only JVM a Java programmer can also assign the priorities of a thread explicitly in a Java program.

Setter & Getter Method of Thread Priority

- **public final int getPriority():**
- The java.lang.Thread.getPriority() method returns the priority of the given thread.
- **public final void setPriority(int newPriority):**
- The java.lang.Thread.setPriority() method updates or assign the priority of the thread to newPriority.
- The method throws IllegalArgumentException if the value new Priority goes out of the range, which is 1 (minimum) to 10 (maximum).

3 constants defined in Thread class:

- **public static int MIN_PRIORITY**
- **public static int NORM_PRIORITY**
- **public static int MAX_PRIORITY**

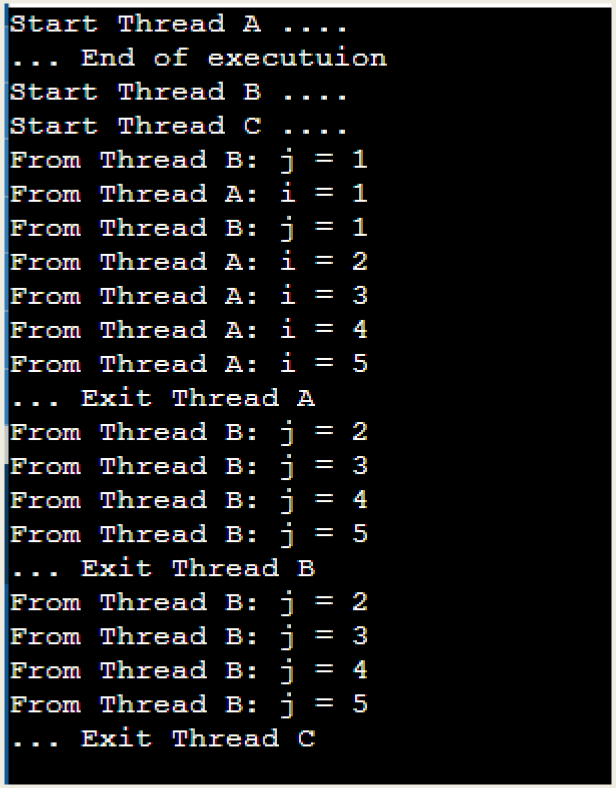
Default priority of a thread is **5 (NORM_PRIORITY).**

The **value of MIN_PRIORITY is 1** and the value of **MAX_PRIORITY is 10**.

```
class ThreadA extends Thread{
        public void run() {
                System.out.println("Start Thread A ....");
                for(int i = 1; i <= 5; i++) {
                        System.out.println("From Thread A: i = "+ i);
                }
                System.out.println("... Exit Thread A");
        }
}

class ThreadB extends Thread{
public void run() {
        System.out.println("Start Thread B ....");
        for(int j = 1; j <= 5; j++) {
                System.out.println("From Thread B: j = "+ j);
        }
        System.out.println("... Exit Thread B");
        }
}
```

```
class ThreadC extends Thread{
    public void run() {
    System.out.println("Start Thread C ....");
    for(int k = 1; k <= 5; k++) {
        System.out.println("From Thread B: j = "+ k);
    }
    System.out.println("... Exit Thread C");
    }
}
class  Priority{
    public static void main (String args[]) {
        ThreadA t1 = new ThreadA();
        ThreadB t2 = new ThreadB();
        ThreadC t3 = new ThreadC();
        t3.setPriority(Thread.MAX_PRIORITY);
        t2.setPriority(t2.getPriority() + 1);
        t1.setPriority(Thread.MIN_PRIORITY);
        t1.start(); t2.start(); t3.start();
        System.out.println("... End of executuion ");
    }
}
```



```
Start Thread A ....
... End of executuion
Start Thread B ....
Start Thread C ....
From Thread B: j = 1
From Thread A: i = 1
From Thread B: j = 1
From Thread A: i = 2
From Thread A: i = 3
From Thread A: i = 4
From Thread A: i = 5
... Exit Thread A
From Thread B: j = 2
From Thread B: j = 3
From Thread B: j = 4
From Thread B: j = 5
... Exit Thread B
From Thread B: j = 2
From Thread B: j = 3
From Thread B: j = 4
From Thread B: j = 5
... Exit Thread C
```

Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.

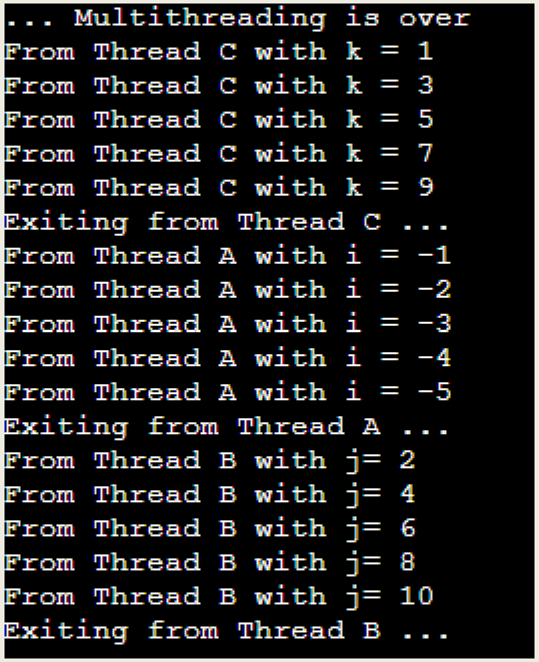
```
class ThreadA extends Thread{
     public void run( ) {
        for(int i = 1; i <= 5; i++) {
           System.out.println("From Thread A with i =
"+ -1*i);
        }
        System.out.println("Exiting from Thread A ...");
     }
}

class ThreadB extends Thread {
    public void run( ) {
       for(int j = 1; j <= 5; j++) {
         System.out.println("From Thread B with j=
"+2* j);
       }
       System.out.println("Exiting from Thread B ...");
      }
}
```

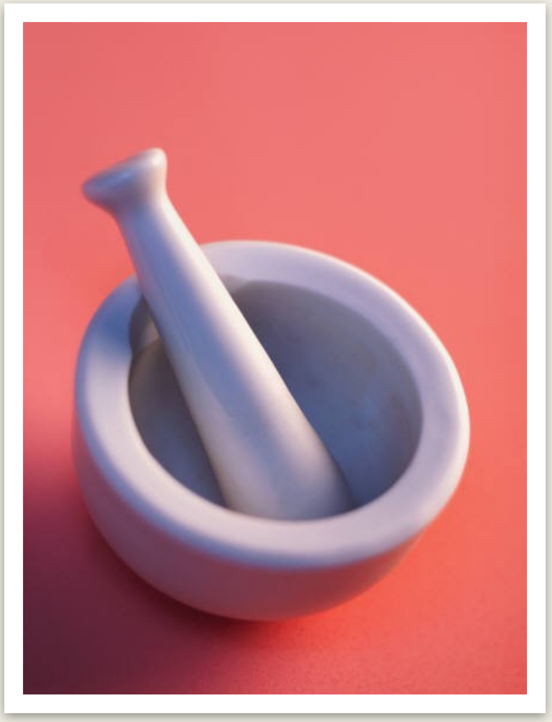
```
class ThreadC extends Thread{
     public void run( ) {
         for(int k = 1; k <= 5; k++) {
             System.out.println("From Thread C with k = "+ (2*k-1));
         }
         System.out.println("Exiting from Thread C ...");
     }
}
public class Multithread {
    public static void main(String args[]) {
         ThreadA a = new ThreadA();
         ThreadB b = new ThreadB();
         ThreadC c = new ThreadC();
         a.start();
         b.start();
         c.start();
         System.out.println("... Multithreading is over ");
    }
}
```

Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.



```
... Multithreading is over
From Thread C with k = 1
From Thread C with k = 3
From Thread C with k = 5
From Thread C with k = 7
From Thread C with k = 9
Exiting from Thread C ...
From Thread A with i = -1
From Thread A with i = -2
From Thread A with i = -3
From Thread A with i = -4
From Thread A with i = -5
Exiting from Thread A ...
From Thread B with j= 2
From Thread B with j= 4
From Thread B with j= 6
From Thread B with j= 8
From Thread B with j= 10
Exiting from Thread B ...
```



Thank You