JAVA MULTITHREADING

DEFINITIONS

Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of the CPU. Each part of such a program is called a **thread**.
A thread in Java is the direction or path that is taken while a program is being executed. So, threads are light-weight processes within a process.

Threads can be created by using two mechanisms:

1. Extending the Thread class
2. Implementing the Runnable Interface

TYPES OF THREADS

User threads

These are high-priority threads. The JVM will wait for any user thread to complete its task before terminating it.

Daemon threads

They are low-priority threads whose only role is to provide services to user threads.

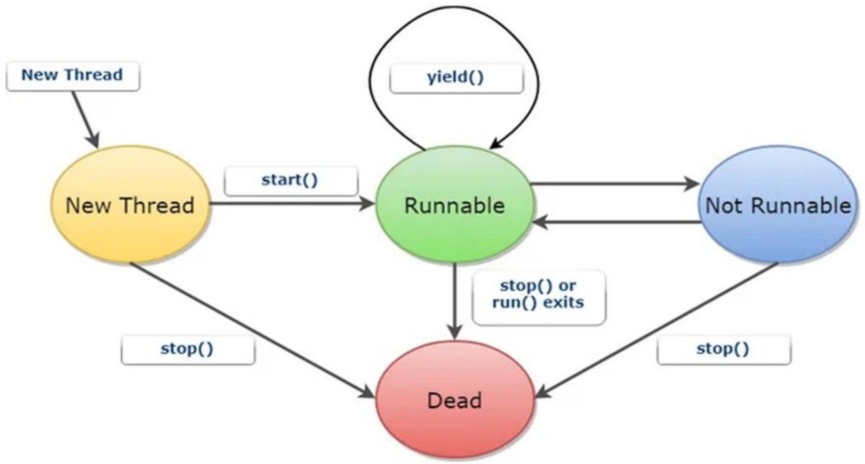
ADVANTAGES OF MULTITHREADING

1. The users are not blocked because threads are independent, and we can perform multiple operations at times

2. As such the threads are independent, the other threads won't get affected if one thread meets an exception.

THE LIFE CYCLE OF A THREAD

- **New Thread:** A new thread begins its life cycle in the new state. The process remains in this condition until the program starts the thread.
- **Runnable:** As soon as the new thread starts, the thread status becomes Runnable. At this stage, a thread is considered to execute its function or working.
- **Not Runnable:** A Runnable thread when entered the time of waiting for the state for a specific interval of time. That time, the thread is not in Runnable condition.
- **Dead / Terminated:** The Runnable thread enters the end stage when it completes its tasks.



Multithreading as a form of Multitasking

- **Multitasking** is the feature that allows your computer to run two or more programs at the same time.
- **Multithreading** is basically a thread-based form of multitasking.
- Thread-based multitasking deals with the concurrent execution of pieces of the same program.
- A multithreaded program contains two or more parts that can run concurrently.
- Each part of such a program is called a **thread**, and each thread defines a **separate path of execution**.

Process-based Multitasking	Thread-based Multitasking
• Your OS runs multiple processes while are multiple programs. • If process based, those are independent from each other. • You can kill any process at any time without affecting others.	• These threads belong to same main process that run in an Operating system. • Within same process, we divide into multiple tasks.

IMPLEMENTING RUNNABLE INTERFACE vs THREAD CLASS IN JAVA

If we extend the **Thread class**, our class cannot extend any other class because Java doesn't support multiple inheritance. But, if we implement the **Runnable interface**, our class can still extend other base classes.

We can achieve basic functionality of a thread by extending **Thread class** because it provides some inbuilt methods like yield(), interrupt() etc. that are not available in **Runnable interface**.

Using runnable will give you an object that can be shared amongst multiple threads.

Thread creation by extending the Thread class

We create a class that extends the **java.lang.Thread** class.
This class overrides the run() method available in the Thread class.
A thread begins its life inside run() method.
We create an object of our new class and call start() method to start the execution of a thread.
Start() invokes the run() method on the Thread object.

```java
// Java code for thread creation by extending
// the Thread class
class MultithreadingDemo extends Thread {
    public void run()
    {
        try {
            // Displaying the thread that is running
            System.out.println(
                "Thread " + Thread.currentThread().getId()
                + " is running");
        }
        catch (Exception e) {
            // Throwing an exception
            System.out.println("Exception is caught");
        }
    }
}

// Main Class
public class Multithread {
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i = 0; i < n; i++) {
            MultithreadingDemo object
                = new MultithreadingDemo();
            object.start();
        }
    }
```

Output

```
Thread 15 is running
Thread 14 is running
Thread 16 is running
Thread 12 is running
Thread 11 is running
Thread 13 is running
Thread 18 is running
Thread 17 is running
```

Thread creation by implementing the Runnable Interface

We create a new class which implements java.lang.
Runnable interface and override run() method.
Then we instantiate a Thread object and call start() method on this object.

```java
// Java code for thread creation by implementing
// the Runnable Interface
class MultithreadingDemo implements Runnable {
    public void run()
    {
        try {
            // Displaying the thread that is running
            System.out.println(
                "Thread " + Thread.currentThread().getId()
                + " is running");
        }
        catch (Exception e) {
            // Throwing an exception
            System.out.println("Exception is caught");
        }
    }
}

// Main Class
class Multithread {
    public static void main(String[] args)
    {
        int n = 8; // Number of threads
        for (int i = 0; i < n; i++) {
            Thread object
                = new Thread(new MultithreadingDemo());
            object.start();
        }
    }
```

Output

```
Thread 13 is running
Thread 11 is running
Thread 12 is running
Thread 15 is running
Thread 14 is running
Thread 18 is running
Thread 17 is running
Thread 16 is running
```

CREATION OF A THREAD

1. Implementing the Runnable Interface

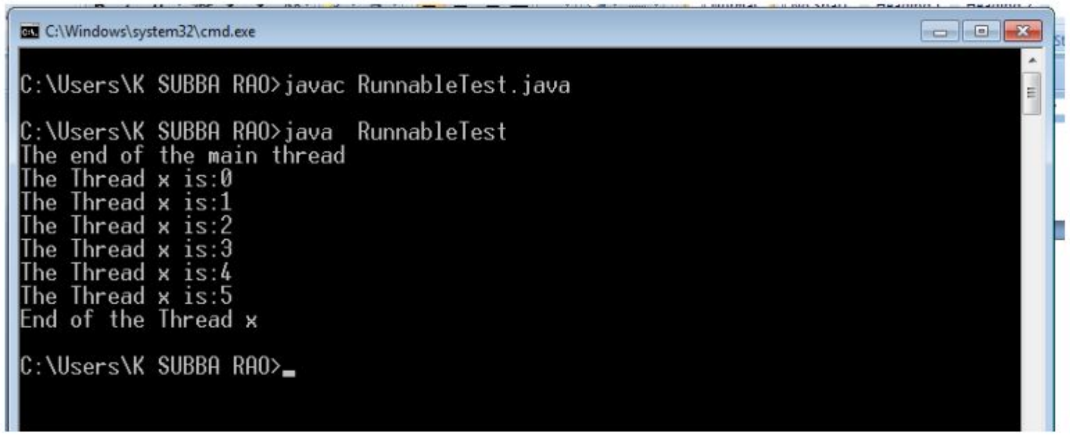
Example program:

Runnable.java

```
class x implements Runnable
{ //1 STEP
        public void run()
        { //2 STEP

                for(int i=0;i<=5;i++)
                System.out.println("The Thread x is:"+i);
                System.out.println("End of the Thread x");
        }
}
class RunnableTest
{
        public static void main(String args[])
        {
                x r=new x();
                Thread threadx=new Thread(r);
                threadx.start();
                System.out.println("The end of the main thread");
```

Output:



```
C:\Users\K SUBBA RAO>javac RunnableTest.java

C:\Users\K SUBBA RAO>java  RunnableTest
The end of the main thread
The Thread x is:0
The Thread x is:1
The Thread x is:2
The Thread x is:3
The Thread x is:4
The Thread x is:5
End of the Thread x

C:\Users\K SUBBA RAO>_
```

CREATION OF A THREAD

2. Extending the thread class

Example program:

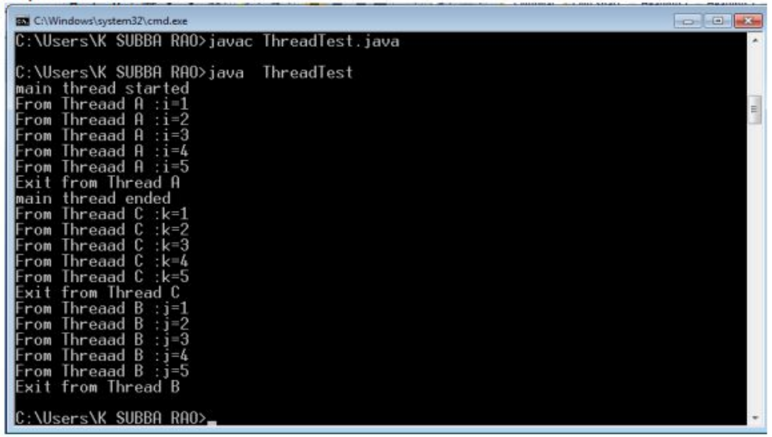
ThreadTest.java

```
import java.io.*;
import java.lang.*;

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

class B extends Thread
{
        public void run()
        {
                for(int j=1;j<=5;j++)
                {
                        System.out.println("From Threaad B :j="+j);
                }
                System.out.println("Exit from Thread B");
        }
}
class C extends Thread
{
        public void run()
        {
                for(int k=1;k<=5;k++)
                {
                        System.out.println("From Threaad C :k="+k);
                }
                System.out.println("Exit from Thread C");
        }
}
class ThreadTest
{
        public static void main(String args[])
        {
                System.out.println("main thread started");
                A a=new A();
                a.start();
                B b=new B();
                b.start();
                C c=new C();
                c.start();
        System.out.println("main thread ended");
        }
}
```

output: First Run



```
C:\Users\K SUBBA RAO>javac ThreadTest.java

C:\Users\K SUBBA RAO>java  ThreadTest
main thread started
From Threaad A :i=1
From Threaad A :i=2
From Threaad A :i=3
From Threaad A :i=4
From Threaad A :i=5
Exit from Thread A
main thread ended
From Threaad C :k=1
From Threaad C :k=2
From Threaad C :k=3
From Threaad C :k=4
From Threaad C :k=5
Exit from Thread C
From Threaad B :j=1
From Threaad B :j=2
From Threaad B :j=3
From Threaad B :j=4
From Threaad B :j=5
Exit from Thread B

C:\Users\K SUBBA RAO>_
```

Second Run: Produces different out put in the second run, because of the processor switching from one thread to other.

THANKS!

Group Members:

Nicole Owens
Neema Karanu
Alex Wafula
Ruby Mbete