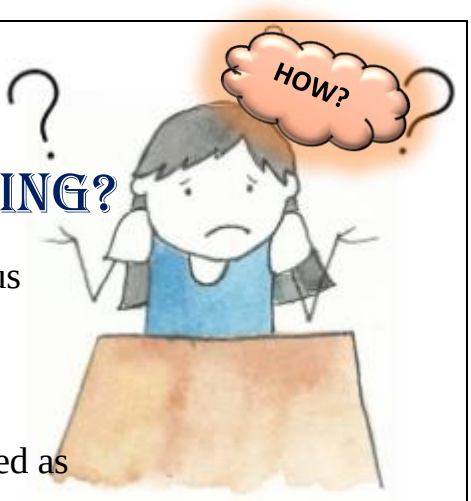
HOW TO ACHIEVE MULTITHREADING?



Before understanding how to achieve multithreading, let us Understand the concept of threads with the help of code.

The stack which is **automatically created by java** is called as **Main stack.**

To execute the contents of main stack, **thread scheduler automatically creates a line of execution called as main thread.**

A thread is a part of a process or is the path followed when executing a program.

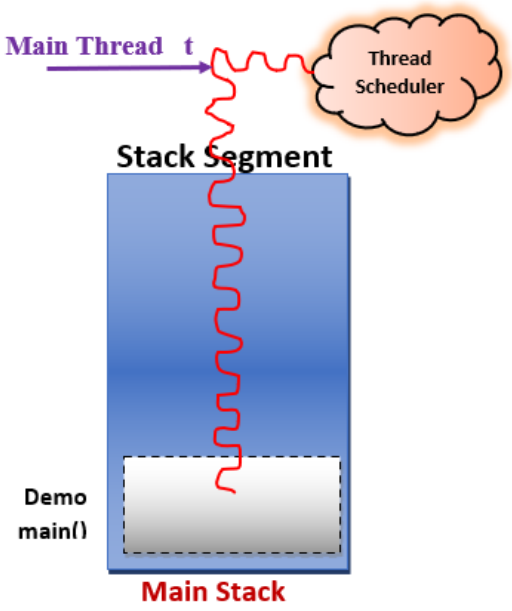
Thread scheduler in java is a software **that decides which thread should run**.

Only one thread at a time can run in a single process.

The code below consists **of main thread only** and hence it is a **single threaded program.**

Code Segment

```
class Demo
{
    public static void main(String[ ] args)
    {
        Thread t = Thread.currentThread();
        System.out.println(t);
    }
}
```



OUTPUT:

Thread [main ,5, main]

Thread Name **Priority** **Method Name**

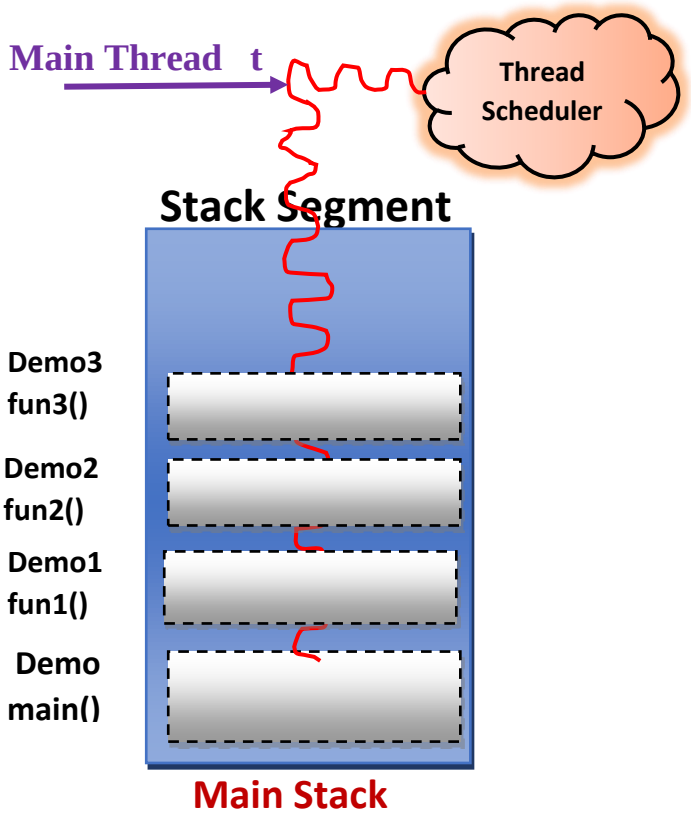
Can we achieve multithreading using multiple methods?

Let us write code with three different activities in three different methods and look at its output to understand this.

```
import java.util.Scanner;
class Demo1
{
    void fun1()
    {
        System.out.println("Addition task started");
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the first number");
        int a = scan.nextInt();
        System.out.println("Enter the second number");
        int b = scan.nextInt();
        int c = a+b;
        System.out.println(c);
        System.out.println("Addition task completed");
    }
}
class Demo2
{
    void fun2() throws Exception
    {
        System.out.println("Character printing started");
        for(int i=65;i<=75;i++)
        {
            System.out.println((char)i);
            Thread.sleep(1000);
        }
        System.out.println("Character printing completed");
    }
}
class Demo3
{
    void fun3() throws Exception
    {
        System.out.println("Number printing started");
        for(int i=1;i<=10;i++)
        {
            System.out.println(i);
            Thread.sleep(1000);
        }
        System.out.println("Number printing Completed");
    }
}
```

```
class Demo
{
    public static void main(String[ ] args) throws Exception
    {
        Demo1 d1 = new Demo1();
        Demo2 d2 = new Demo2();
        Demo3 d3 = new Demo3();
        d1.fun1();
        d2.fun2();
        d3.fun3();
    }
}
```

Let us see how stack segment of this code looks like.

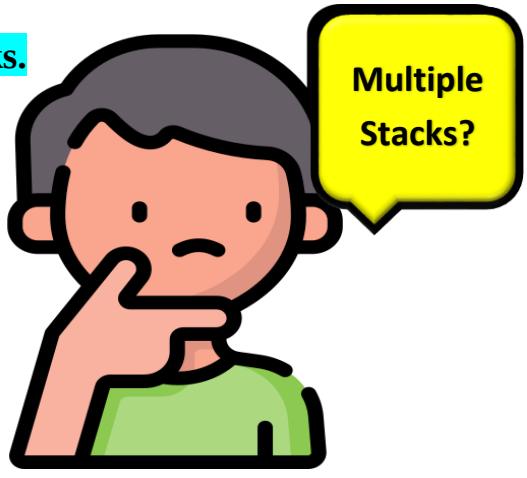


The stack segment above consists **of main stack** only which in turn consists of **main thread only** and hence it is a **single threaded program.**

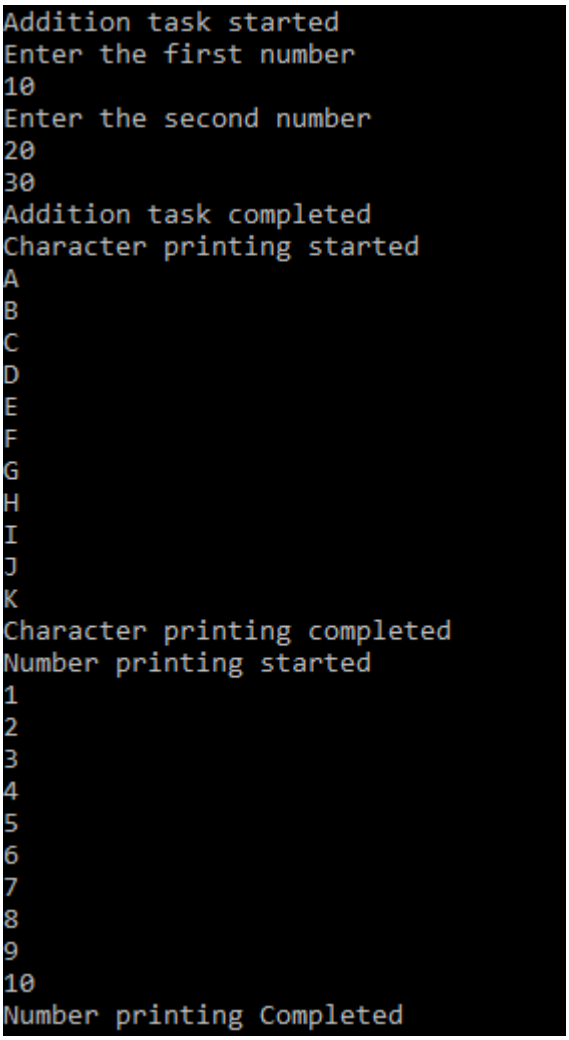
In conclusion, multithreading cannot be achieved using multiple methods.

Then how do we achieve it???

We achieve by creating multiple-stacks.



OUTPUT:

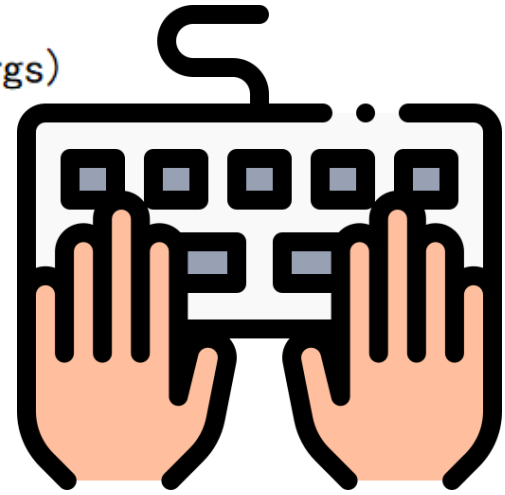


Multithreading can be achieved by creating multiple stacks
Extra stack can be created by creating an object of thread class.
For every new stack, the thread scheduler will create a new thread.
The independent activities which must be concurrently executed should be placed within run().
The programmer must never directly call run().
It must always be indirectly called using start().
When start() is called, a new thread is created and code inside run() method is executed in new thread while if you call run() directly no new thread is created and code inside run() will execute on current thread or main thread.

Let us write code to understand this in detail


```
import java.util.Scanner;
class Demo1 extends Thread
{
    public void run()
    {
        System.out.println("Addition task started");
        Scanner scan = new Scanner(System.in);
        System.out.println("Enter the first number");
        int a = scan.nextInt();
        System.out.println("Enter the second number");
        int b = scan.nextInt();
        int c = a+b;
        System.out.println(c);
        System.out.println("Addition task completed");
    }
}
class Demo2 extends Thread
{
    public void run()
    {
            System.out.println("Character printing started");
            for(int i=65;i<=75;i++)
            {
                System.out.println((char)i);
                try
                {
                    Thread.sleep(4000);
                }
                catch (Exception e)
                {
                    System.out.println("Some problem occured");
                }
            }
            System.out.println("Character printing completed");
    }
}
class Demo3 extends Thread
{
    public void run()
    {
            System.out.println("Number printing started");
```

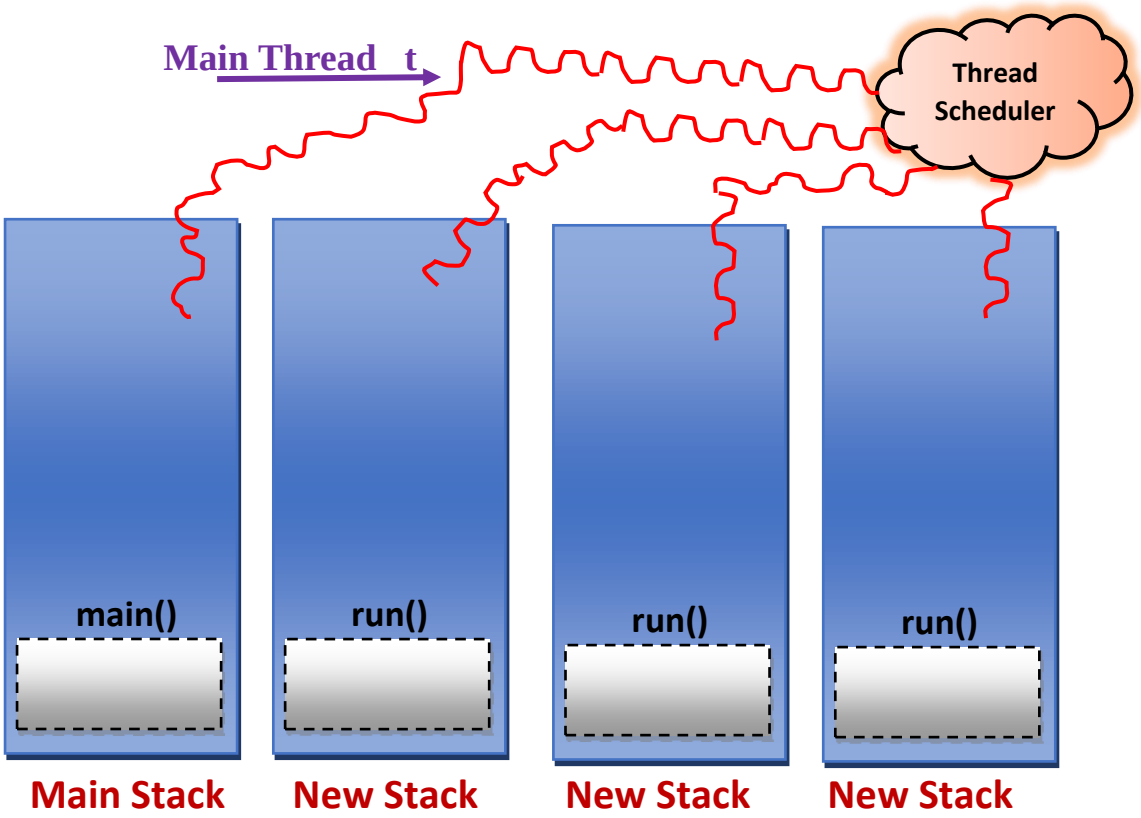
```
            for(int i=1;i<=10;i++)
            {
                System.out.println(i);
                try
                {
                    Thread.sleep(4000);
                }
                catch (Exception e)
                {
                    System.out.println("Some problem occured");
                }
            }
            System.out.println("Number printing Completed");
        }
    }

    class Demo
    {
        public static void main(String[ ] args)
        {
            Demo1 d1 = new Demo1();
            Demo2 d2 = new Demo2();
            Demo3 d3 = new Demo3();
            d1.start();
            d2.start();
            d3.start();
        }
    }
```



In the above code, all the three activities that is addition, printing alphabets and printing numbers are written inside run() and every class is extending thread class. Since every class is extending thread class, when an object is created of these thread classes, new stack gets created and now thread scheduler creates new thread for each of these stacks due to which concurrent execution takes place and CPU time is utilised efficiently.

Let us look at its output and stack segment.

STACK SEGMENT:



OUTPUT:

```
Addition task started
Number printing started
Character printing started
A
1
Enter the first number
2
Enter the second number
B
2
C
3
D
4
E
5
F
6
400
402
Addition task completed
G
7
H
8
I
9
J
10
K
Number printing Completed
Character printing completed
```