

By implementing the Runnable interface:

```

MyThread-----> Thread-----> Runnable
      Extends                already implemented

```

```
MyRunnable -----> Runnable
           Implements
```

How we can define a thread through Runnable

```
class UsingRunnable implements Runnable
{
    public void run()
    {
        System.out.println("thread is running...");
    }
    public static void main(String args[]){
        UsingRunnable r=new UsingRunnable();
        Thread t =new Thread(r);
        t.start();
    } }

```

```
C:\Windows\System32\cmd.exe

F:\Java Code>javac UsingRunnable.java

F:\Java Code>java UsingRunnable
thread is running...
```

```
class UsingRunnable implements Runnable
{
    public void run()
    {
for(int i=1;i<=10;i++)
    {
System.out.println("child thread is running..." +i);
    }
    }
    public static void main(String args[])
    {
        UsingRunnable r=new UsingRunnable();
        Thread t =new Thread(r);
        t.start();
for(int i=11;i<=20;i++)
    {
System.out.println("main thread is running..." +i);
    }
    } }
```

```
F:\Java Code>java UsingRunnable  
main thread is running...11  
child thread is running...1  
main thread is running...12  
child thread is running...2  
child thread is running...3  
child thread is running...4  
child thread is running...5  
main thread is running...13  
child thread is running...6  
main thread is running...14  
child thread is running...7  
child thread is running...8  
main thread is running...15  
child thread is running...9  
child thread is running...10  
main thread is running...16  
main thread is running...17  
main thread is running...18  
main thread is running...19  
main thread is running...20
```

Case Study:

```
MyRunnable r=new MyRunnable();  
Thread t1=new Thread();  
Thread t2=new Thread(r);
```

Case 1: t1.start();

A new thread will be created which is responsible for the execution of the Thread class run() method. Which has an empty implementation.

Case 2: t1.run();

No new thread will be created and Thread class run method will be executed just like a normal method call.

Case 3: t2.start();

A new thread will be created which is responsible for the execution of MyRunnable class run method.

Case 4: t2.run();

A new thread won't be created and MyRunnable run() method will be executed just like a normal method call.

Case 5: r.start();

r has no start capability, we will get a compile-time error saying the MyRunnable class doesn't have start() capability.

Case 6: r.run();

no new thread will be created and MyRunnable run() method will be executed like a normal method call.

Which approach is best to define a thread and why?

Second (Recommended): Implement Runnable approach.

Reason:

In the first approach, our class extends the Thread class there is no chance of extending any other class, hence we are missing inheritance benefits.

But in the second approach while implementing a runnable interface we can extend any other class hence we won't miss any inheritance benefits.

Because of the above reason implementing the Runnable interface approach is recommended than extending the Thread class.

Program:

```
class Mythread1 implements Runnable {
    public void run() {
        for(int i=0;i<10;i++) {
            System.out.println("Running Thread1:"+i); }
    } }
class Mythread2 extends Thread {
    public void run() {
        for(int i=10;i<20;i++)
        {
            System.out.println("Running Thread2:"+i); }
    } }
class Runthread {
    public static void main(String arg[]){
        Mythread1 r1=new Mythread1();
        Thread t1=new Thread(r1,"thread1");
        t1.start();
        Mythread2 r2=new Mythread2();
        r2.start();
    }
}
```

C:\Windows\System32\cmd.exe

```
D:\1 Java\Programs>javac Runthread.java
```

```
D:\1 Java\Programs>java Runthread
```

```
Running Thread1:0
```

```
Running Thread1:1
```

```
Running Thread2:10
```

```
Running Thread1:2
```

```
Running Thread1:3
```

```
Running Thread1:4
```

```
Running Thread2:11
```

```
Running Thread2:12
```

```
Running Thread1:5
```

```
Running Thread1:6
```

```
Running Thread1:7
```

```
Running Thread1:8
```

```
Running Thread2:13
```

```
Running Thread2:14
```

```
Running Thread1:9
```

```
Running Thread2:15
```

```
Running Thread2:16
```

```
Running Thread2:17
```

```
Running Thread2:18
```

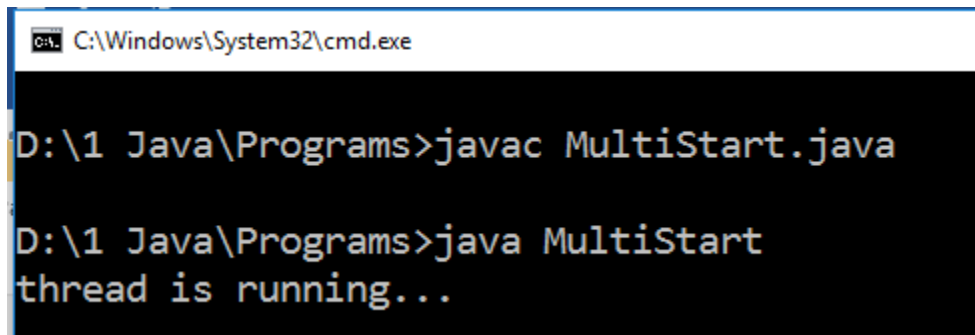
```
Running Thread2:19
```

```
}
```

```

class MultiStart extends Thread{
    public void run()
    {
        System.out.println("thread is running...");
    }
    public static void main(String args[]){
        MultiStart t1=new MultiStart();
        t1.start();
    } }

```



A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\System32\cmd.exe'. The command prompt shows the following sequence of commands and output:

```

D:\1 Java\Programs>javac MultiStart.java

D:\1 Java\Programs>java MultiStart
thread is running...

```

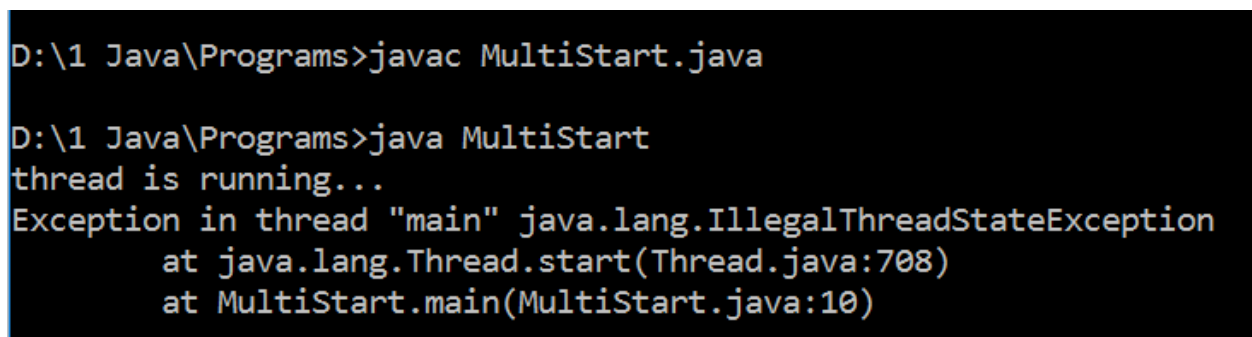
If we start a thread twice:

```

class MultiStart extends Thread{
    public void run()
    {
        System.out.println("thread is running...");
    }
    public static void main(String args[]){
        MultiStart t1=new MultiStart();

        t1.start();
        t1.start();
    }
}

```



A screenshot of a Windows command prompt window. The title bar shows 'C:\Windows\System32\cmd.exe'. The command prompt shows the following sequence of commands and output:

```

D:\1 Java\Programs>javac MultiStart.java

D:\1 Java\Programs>java MultiStart
thread is running...
Exception in thread "main" java.lang.IllegalThreadStateException
    at java.lang.Thread.start(Thread.java:708)
    at MultiStart.main(MultiStart.java:10)

```