UNIT-V: Alternative Paradigm: Concurrent Programming

Faculty In-charge

Mrinmoyee Mukherjee
Assistant Professor (IT Dept.)
email: mrinmoyeemukherjee@sfit.ac.in

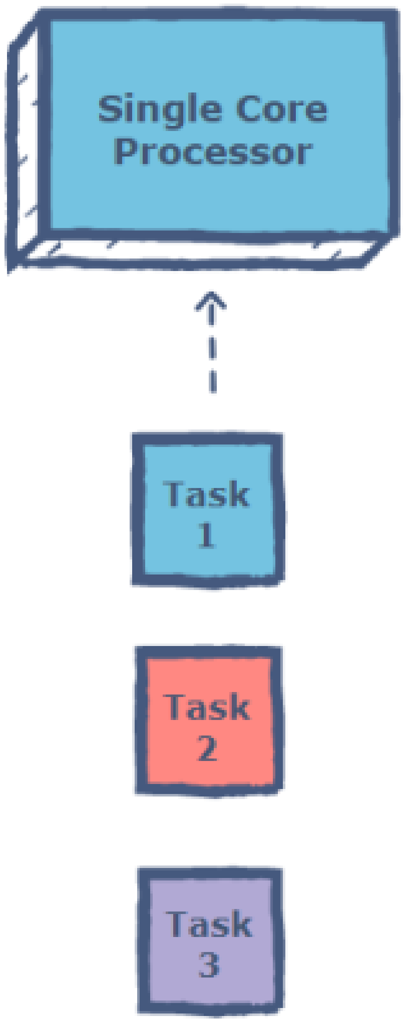
Academic Year: 2023-24

OUTLINE OF UNIT-5

5.0: Concurrent Programming Fundamentals

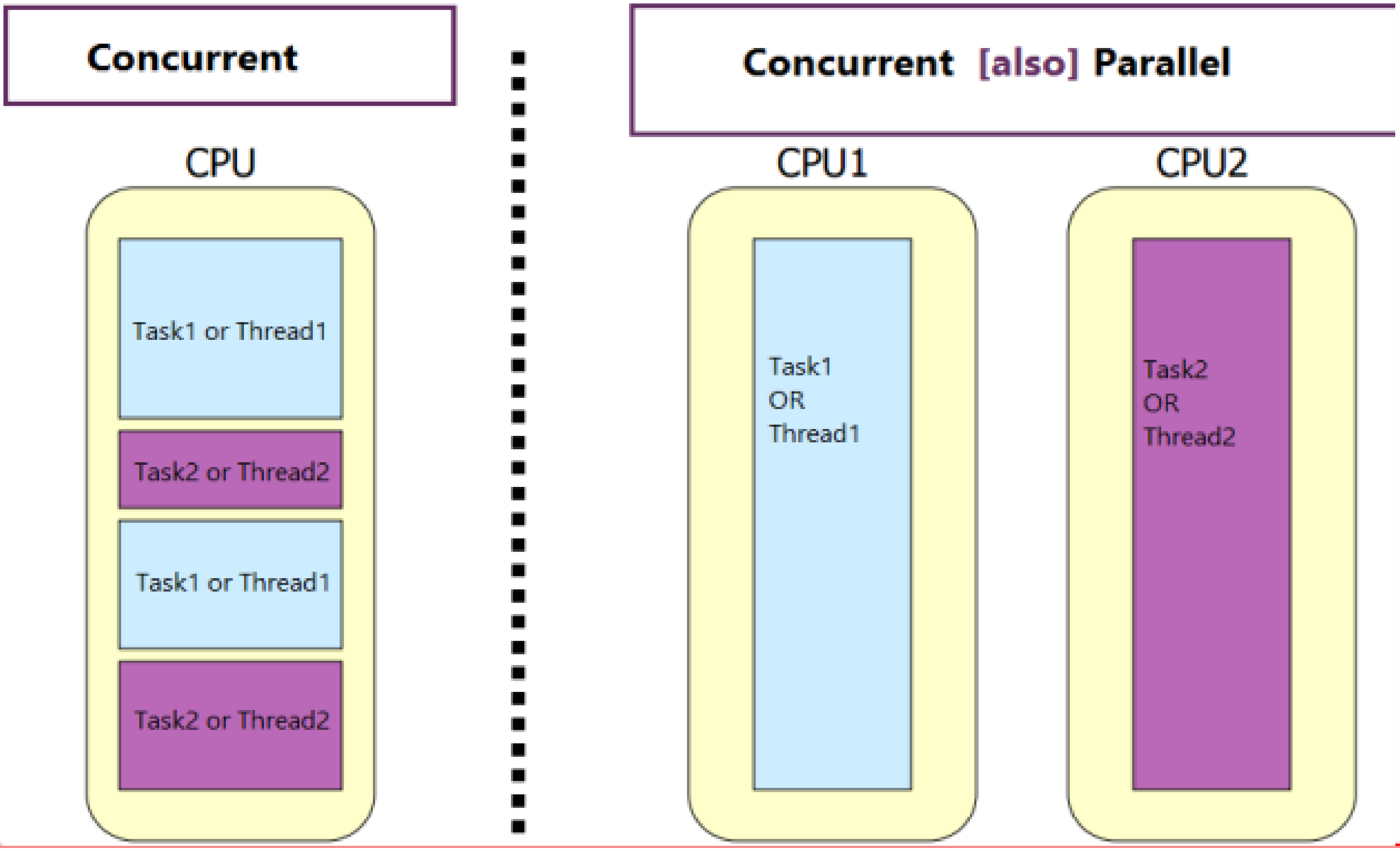
Lect-1: Concurrent Programming Fundamentals

- Concurrency means two or multiple computations are happening at the same time or running in overlapping periods of time. It is to get more work done in less time.
- In programming terms, concurrent programming is a technique in which
  - Two or more processes start
  - Run in an interleaved fashion through switching and
  - Complete in an overlapping time period by managing access to shared resources, e.g. on a single core of CPU.
- Other examples of concurrency:
  - Multiple computers in a network
  - Multiple applications running on one computer
  - Multiple processors in a computer (today, often multiple processor cores on a single chip)
  - Web sites must handle multiple simultaneous users.



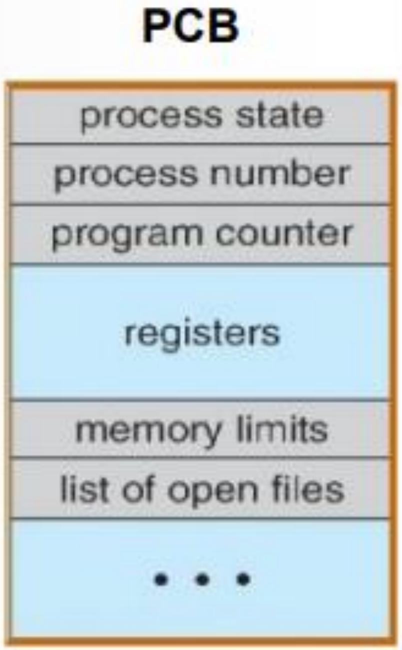
Difference Between Concurrent and Parallel

Concurrency & Parallelism



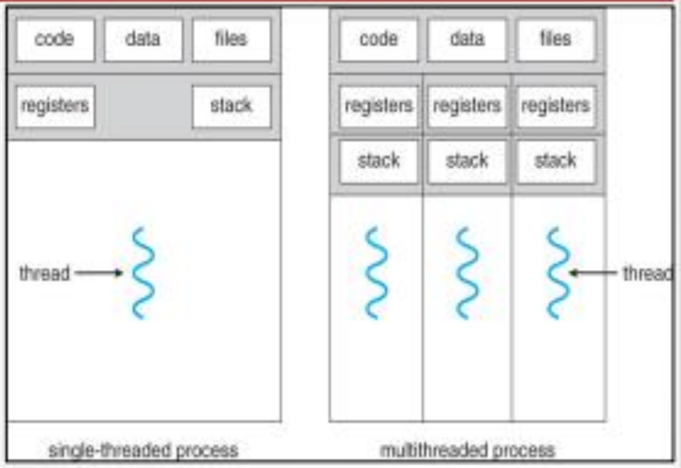
Processes, Threads, Time-slicing

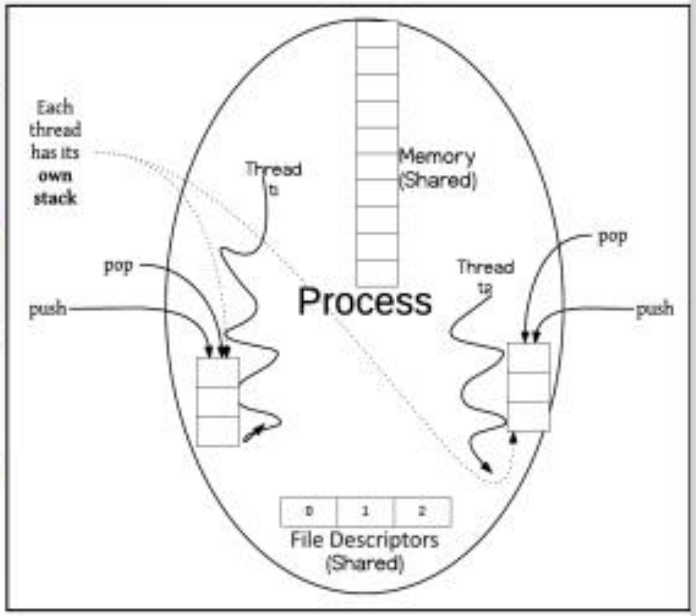
- The concurrent modules themselves come in two different kinds: processes and threads.
- **Process**: A process is an instance of a running program that is isolated from other processes on the same machine.
- In particular, it has its own private section of the machine's memory where process control block (PCB) is stored.
- The process abstraction is a virtual computer. It makes the program feel like it has the entire machine to itself – like a fresh computer has been created, with fresh memory, just to run that program.
- Just like computers connected across a network, processes normally share no memory between them. A process can't access another process's memory or objects at all.
- Sharing memory between processes is possible on most operating system, but it needs special effort.
- By contrast, a new process is automatically ready for message passing, because it is created with standard input & output streams, which are the System.out and System.in streams you've used in Java.



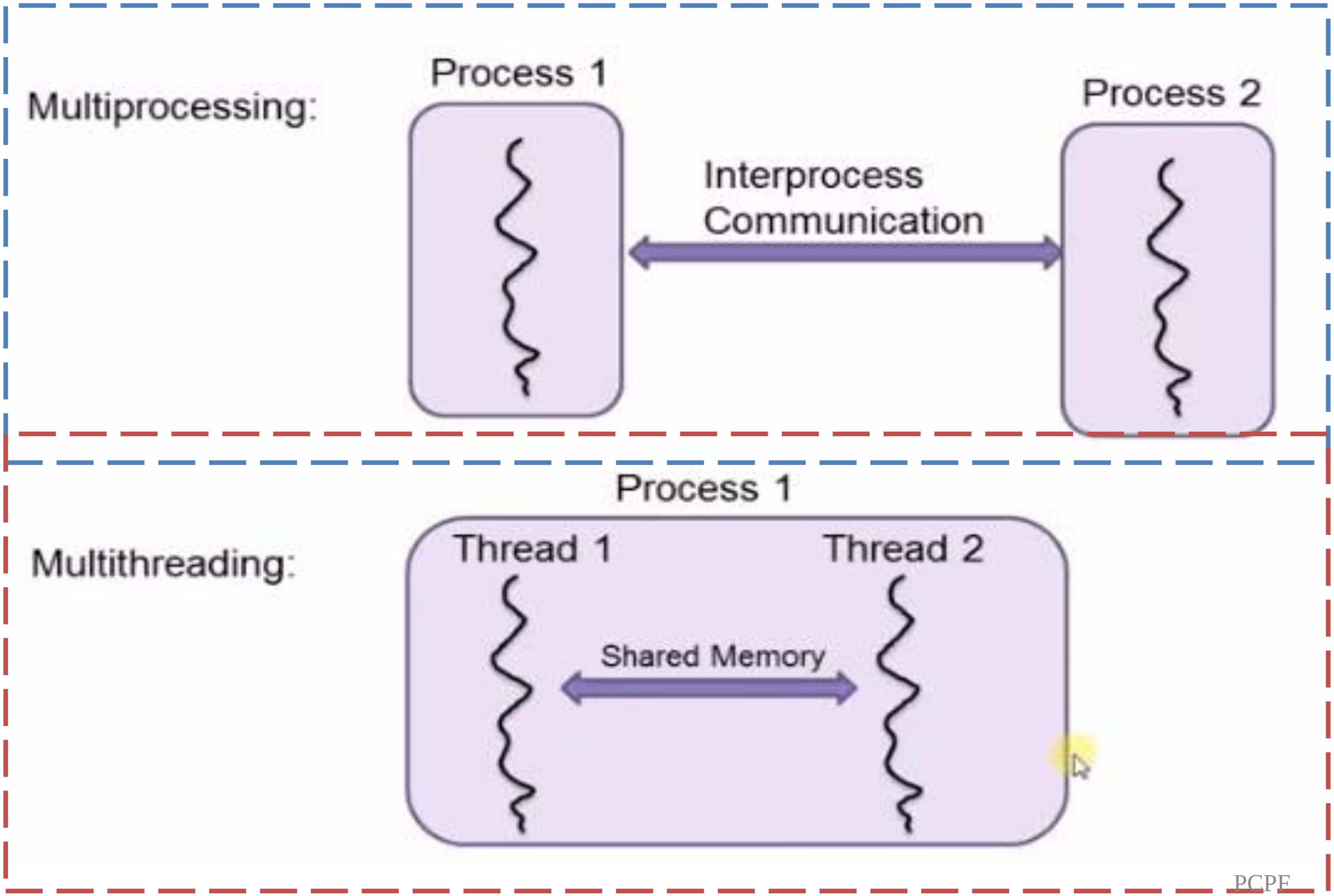
Processes, Threads, Time-slicing

- **Thread**:
- A thread is a locus of control inside a running program. Think of it as a place in the program that is being run.
- Thread is unit of execution within a process.
- Just as a process represents a virtual computer, the thread abstraction represents a virtual processor.
- Making a new thread simulates making a fresh processor inside the virtual computer represented by the process.
- The new thread runs the same program and shares the same memory as other threads in process.
- Threads are automatically ready for shared memory, because threads share all the memory in the process.
- It needs special effort to get "thread-local" memory that's private to a single thread.
- It's also necessary to set up message-passing explicitly, by creating and using queue data structures.

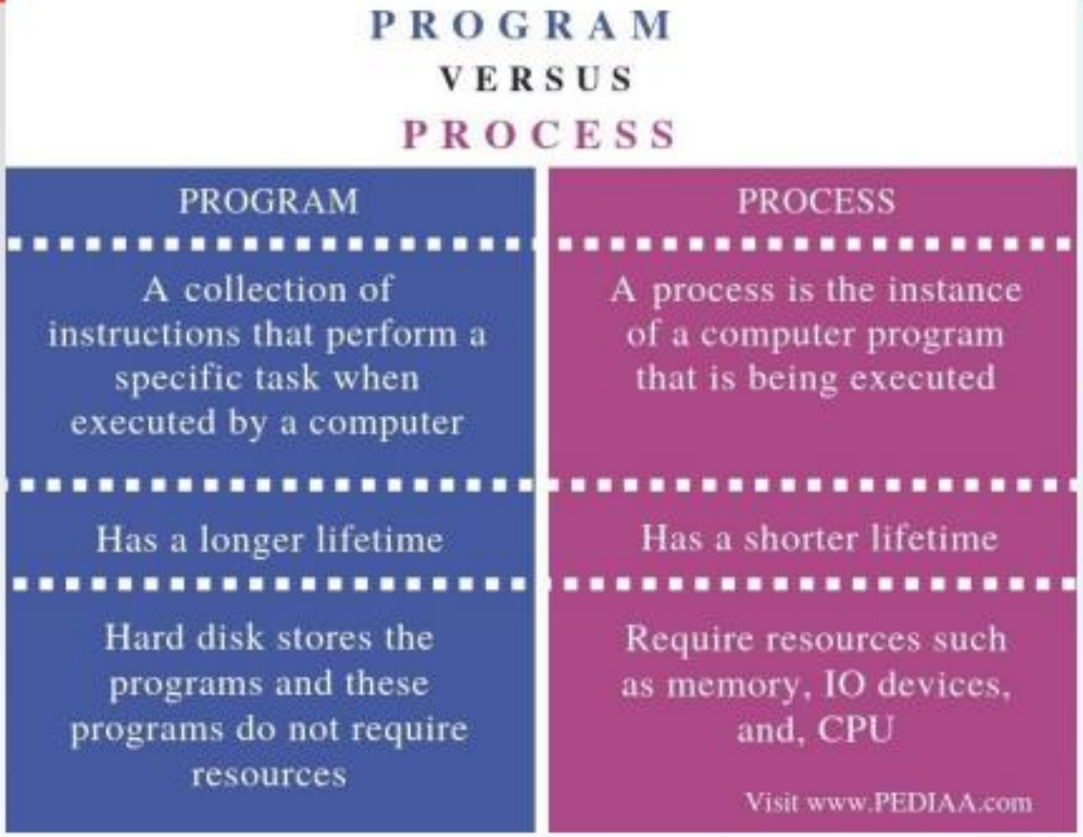




Process vs. Threads



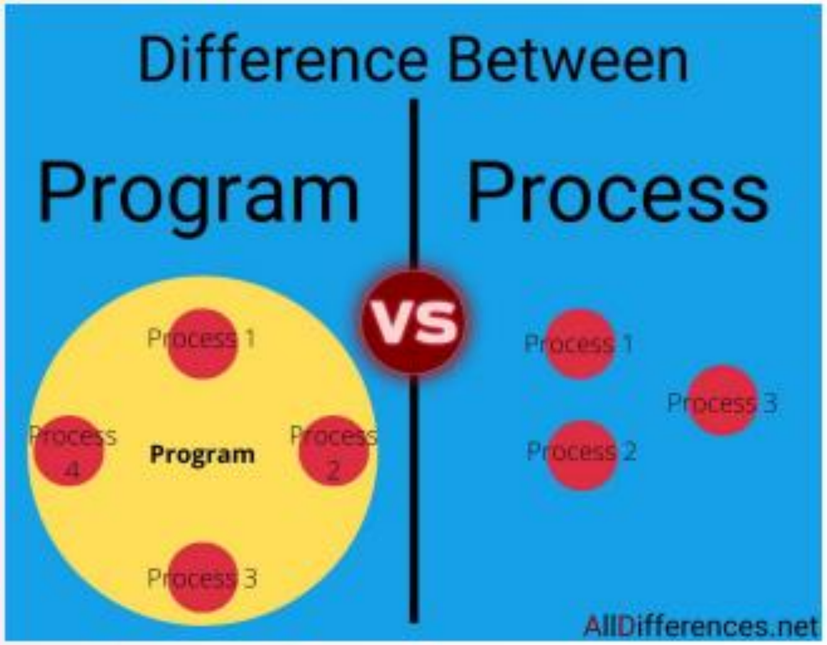
Program Vs Process Vs Thread

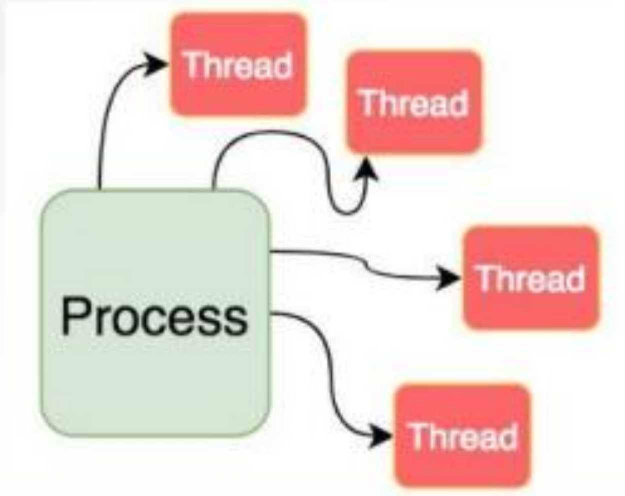


PROGRAM
VERSUS
PROCESS

PROGRAM	PROCESS
A collection of instructions that perform a specific task when executed by a computer	A process is the instance of a computer program that is being executed
Has a longer lifetime	Has a shorter lifetime
Hard disk stores the programs and these programs do not require resources	Require resources such as memory, IO devices, and, CPU

Visit www.PEDIAA.com



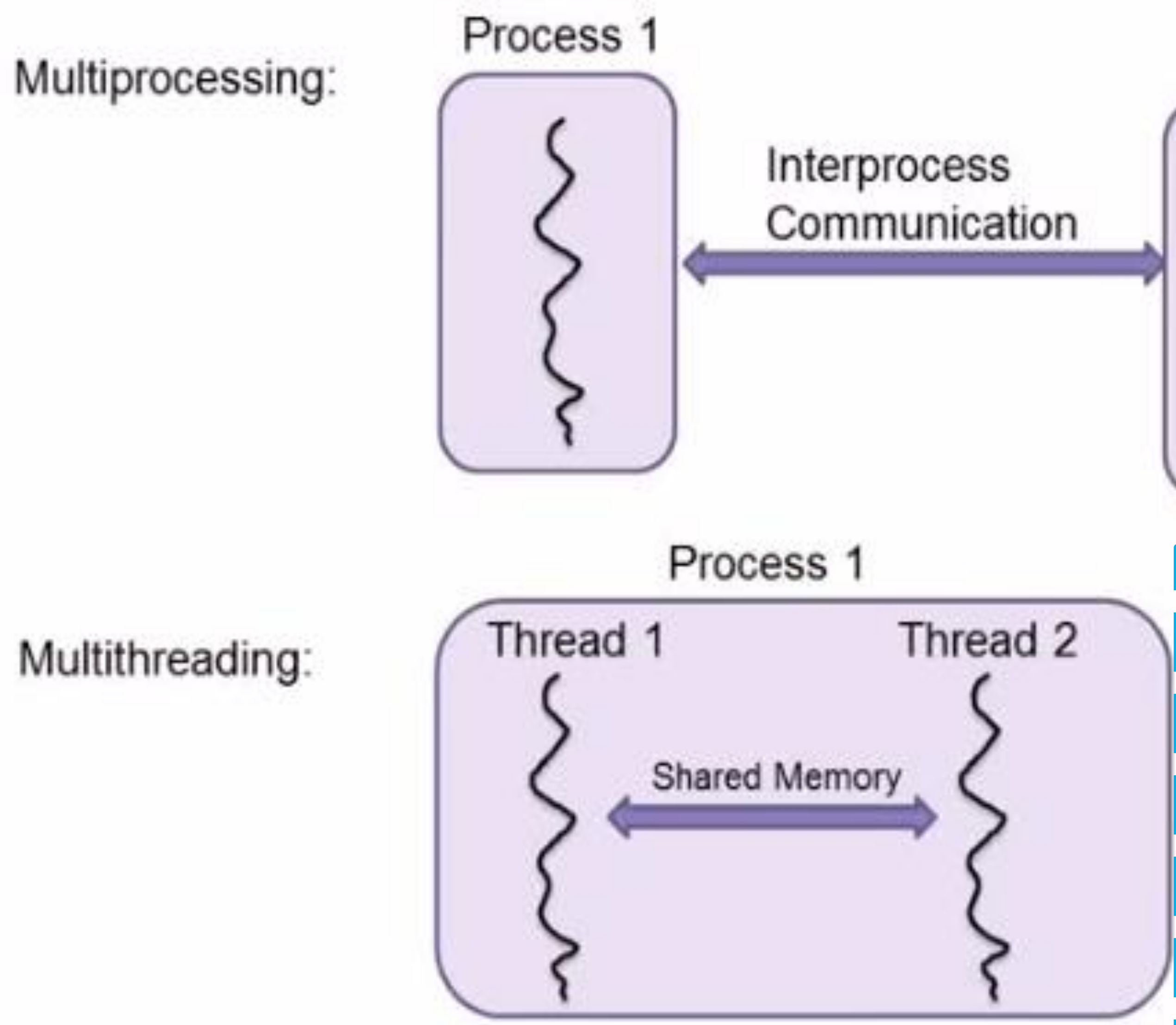


Process

- Process means any program is in execution.
- Process control block contains information about processes for example Process priority, process id, process state, CPU, register, etc.
- A process can creates other processes which are known as **Child Processes**.
- Process takes more time to terminate and it is isolated means it does not share memory with any other process.
- Process is called heavy-weight process

Thread

- One or more **threads** run in the context of the **process**.
- A **thread** is the basic unit to which the operating system allocates processor time.
- A **thread** can execute any part of the **process** code, including parts currently being executed by another **thread**
- **Thread is called light weight process**



Pros:
- Fast to start
- Low overhead

Cons:
- Difficult to implement
- Can't run on distributed system

- In a concurrent program, the term thread defines the present entity which the programmer might think as running concurrently with other threads.
- In a concurrent program, several streams of operations may execute concurrently.
- Each stream of operations executes as it would in a sequential program ***except for the fact that streams can communicate and interfere with one another.***
- Each such sequence of instructions is called a *thread.*
- For this reason, sequential programs are often called *single-threaded* programs.
- When a multi-threaded program executes, the operations in its various threads are interleaved
- The operations for each stream are strictly ordered
- One stream may run very fast while another does not run at all.
- In the absence of fairness guarantees a given thread can starve unless it is the only ``runnable'' thread.

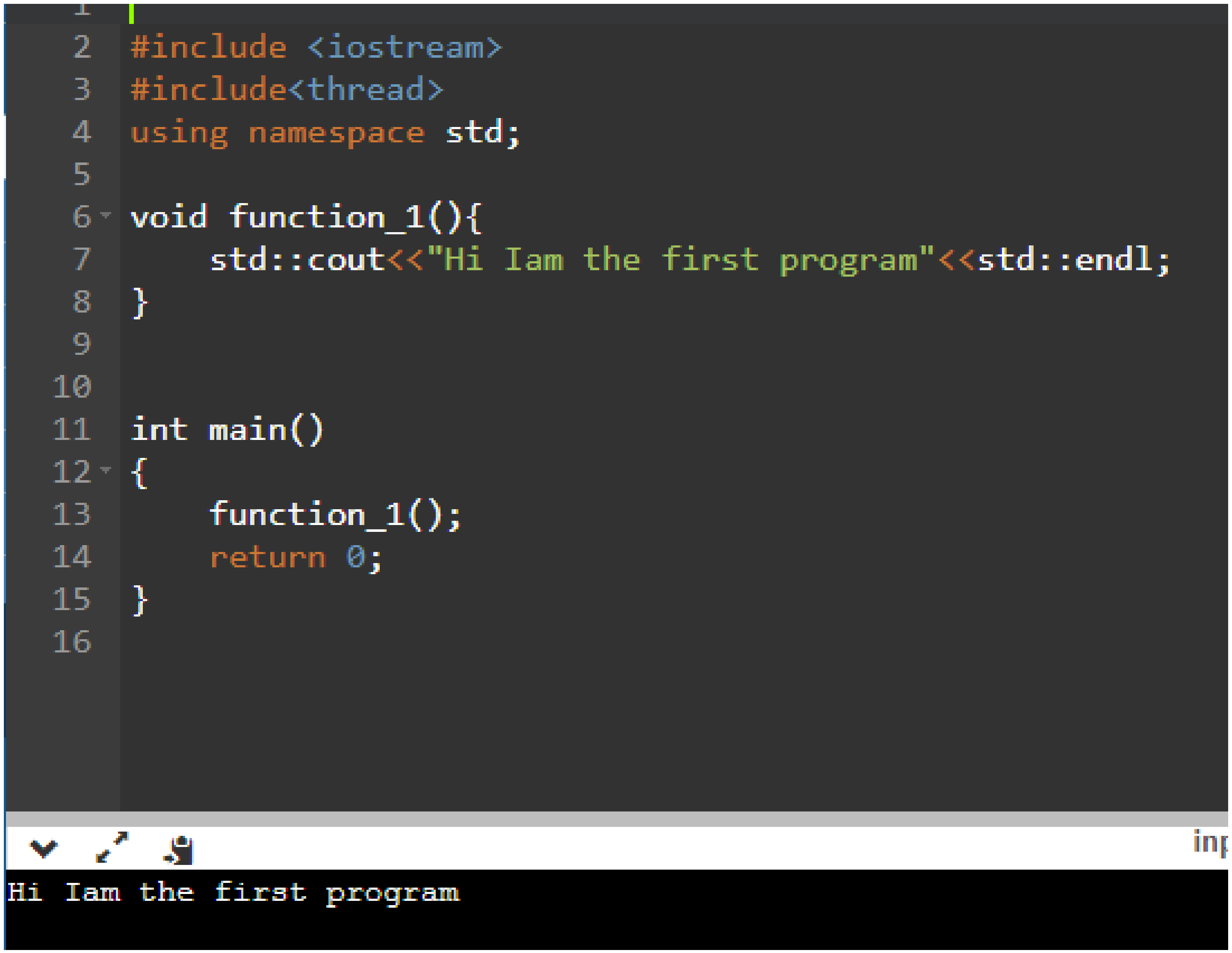
Advantages

1. Run multiple applications at the same time.
2. Without concurrency, every application runs completely before the next one can start running. So concurrency provides an opportunity for everyone
3. Enables better performance by the operating system.

Disadvantages

1. Coordinate with multiple applications via additional mechanisms.
2. In operating systems performance required overheads and complexities to switch among other applications.
3. **Race condition** :Several applications running concurrently can degrade performance. It is an undesirable situation which occurs when a system tends to perform two or more operations at the same time.
4. **Deadlock:** It occurs when more than two processes are blocked and any of processes cannot proceed to execute.
5. **Starvation:** It occurs when a high priority of processes keeps executing and low priority processes get blocked for an indefinite time.

My First Thread....

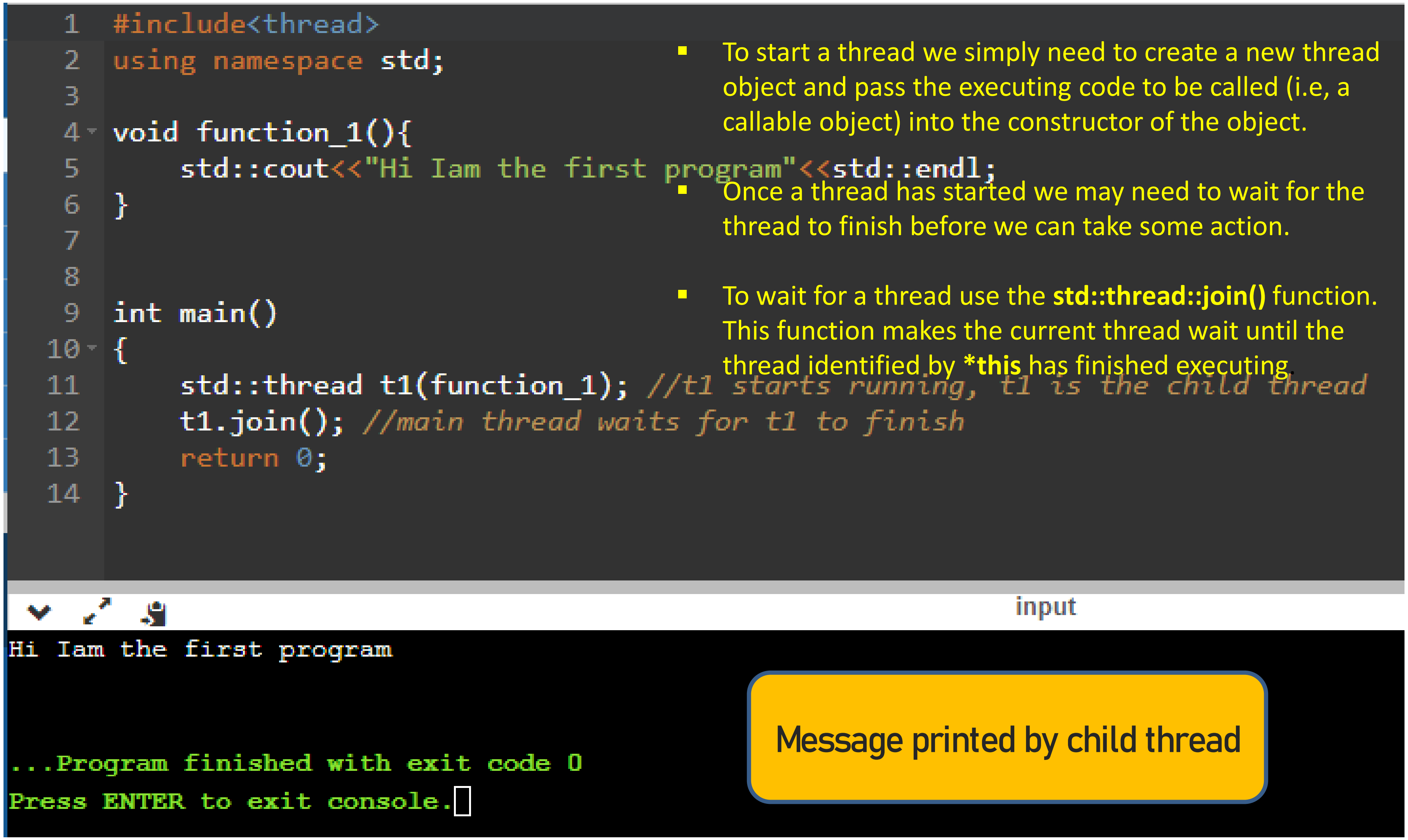


```
#include <iostream>
#include<thread>
using namespace std;

void function_1(){
    std::cout<<"Hi Iam the first program"<<std::endl;
}


int main()
{
    function_1();
    return 0;
}
```

```
Hi Iam the first program
```

However this is not multithreading program… as there is only one thread run

My First Thread….



```
#include<thread>
using namespace std;

void function_1(){
    std::cout<<"Hi Iam the first program"<<std::endl;
}


int main()
{
    std::thread t1(function_1); //t1 starts running, t1 is the child thread
    t1.join(); //main thread waits for t1 to finish
    return 0;
}
```

- To start a thread we simply need to create a new thread object and pass the executing code to be called (i.e, a callable object) into the constructor of the object.
- Once a thread has started we may need to wait for the thread to finish before we can take some action.
- To wait for a thread use the **std::thread::join()** function. This function makes the current thread wait until the thread identified by ***this** has finished executing.

```
Hi Iam the first program


...Program finished with exit code 0
Press ENTER to exit console.
```

Message printed by child thread

Now suppose t1 is a long running thread and my main does not want to wait for it to finish.

5.2: Multithreaded Programs

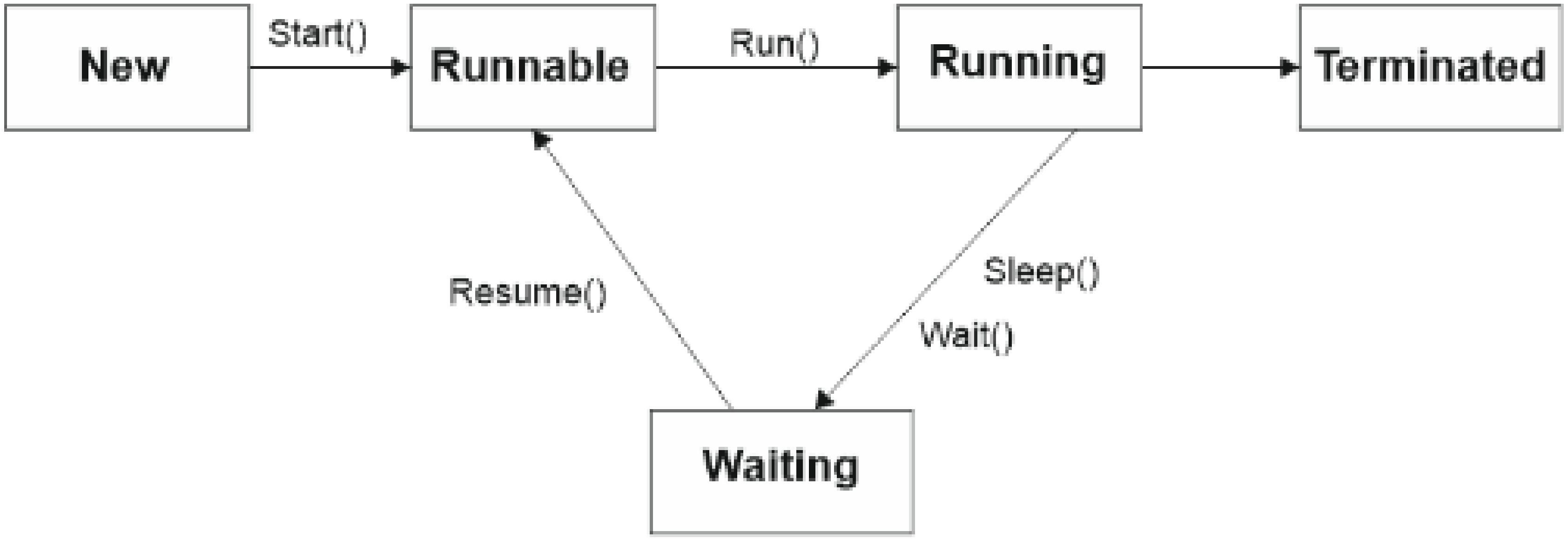
- A thread is the smallest unit of processing.
- They are executed independently, and if they are not synchronized, they can produce wrong output
- Java is a multithreaded programming language. This feature of Java helps in making a Java program run in multiple threads.
- Some of the important properties of threads are as follows:
    - ✓ Threads are lightweight.
    - ✓ Threads run independently.
    - ✓ Threads of a process share the same memory location.
    - ✓ Communication cost of the threads of a process is very low.

In Java, a thread always exists in any one of the following states. These states are: New, Active, Blocked / Waiting, Timed Waiting,Terminated

Thread Life Cycle.....

The life cycle of a thread includes the total processes that it performs from the start of a thread to its termination.

Java Virtual Machine (JVM) controls the lifecycle of a thread in Java.



New: A thread begins its life at the new state, where we create an instance of the Thread class. In this state, the thread is called a born thread. A thread is in this state until a process invokes the start() method.

Active: When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is **runnable**, and the other is **running**.

Runnable: A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. It is the duty of the thread scheduler to provide the thread time to run, i.e., moving the thread the running state.

A program implementing multithreading acquires a fixed slice of time to each individual thread.

Each and every thread runs for a short span of time and when that allocated time slice is over, the thread voluntarily gives up the CPU to the other thread, so that the other threads can also run for their slice of time.

Whenever such a scenario occurs, all those threads that are willing to run, waiting for their turn to run, lie in the runnable state. I

In the runnable state, there is a queue where the threads lie.

Running: When thread scheduler is selected a thread for execution it is called in running state.

Blocked/Waiting: When a thread needs some IO information, it goes in the waiting state. After the waiting state, the thread again moves to the runnable state. Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state.

Example:

For example, a thread (let's say its name is A) may want to print some data from the printer. However, at the same time, the other thread (let's say its name is B) is using the printer to print some data. Therefore, thread A has to wait for thread B to use the printer. Thus, thread A is in the blocked state. A thread in the blocked state is unable to perform any execution and thus never consume any cycle of the Central Processing Unit (CPU). Hence, we can say that thread A remains idle until the thread scheduler reactivates thread A, which is in the waiting or blocked state.

When the main thread invokes the join() method then, it is said that the main thread is in the waiting state. The main thread then waits for the child threads to complete their tasks. When the child threads complete their job, a notification is sent to the main thread, which again moves the thread from waiting to the active state.

Terminated: A thread is in the terminated state if the task of the run()method is completed or terminated.

Thread Creation Syntax

A thread can be created in Java in two ways, either by extending the Thread class or by implementing the runnable interface. Let us learn about both of these.

By Extending the Thread Class

- Thread class provide constructors and methods to create and perform operations on a thread.

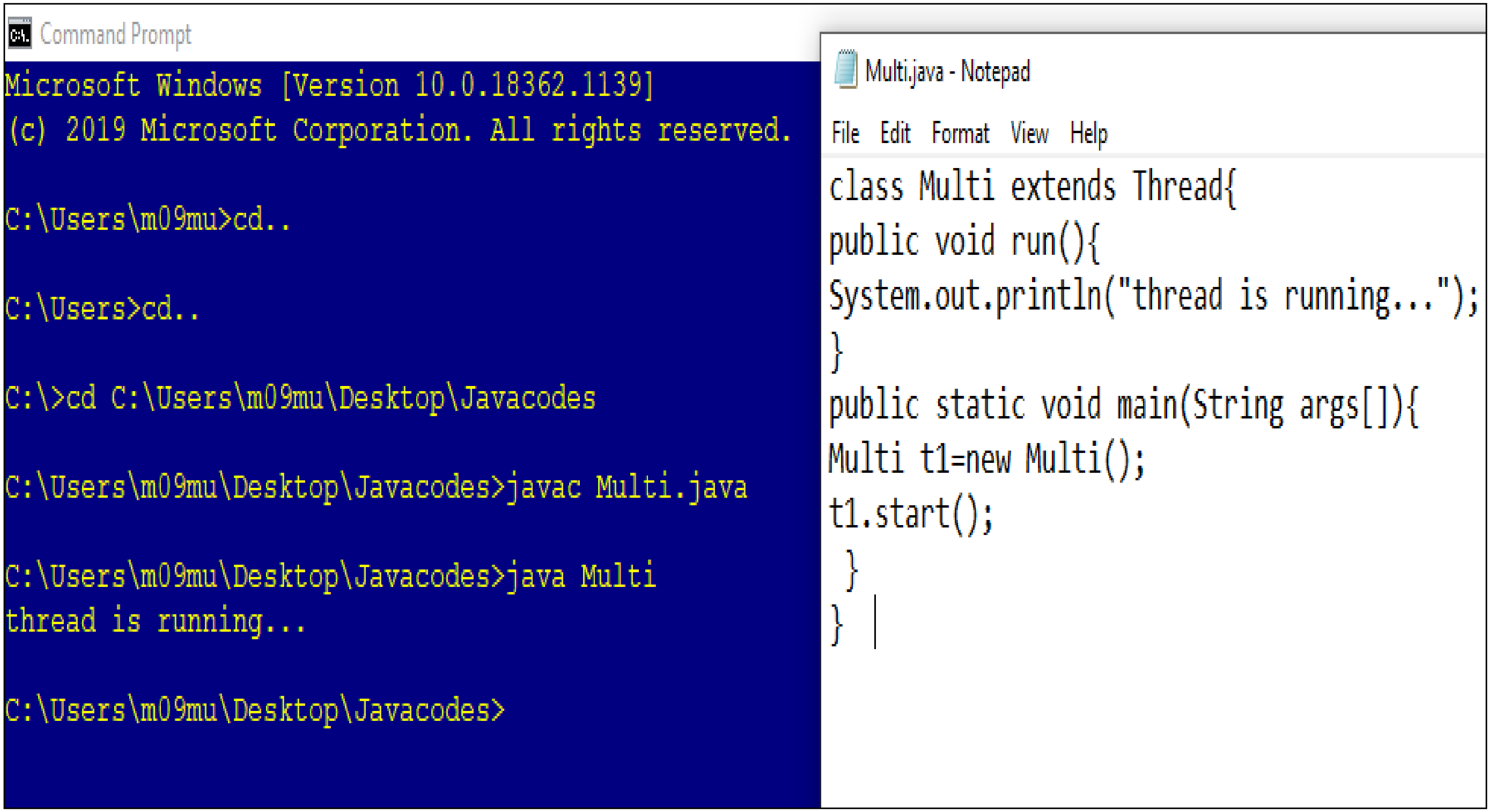
- Thread class extends Object class and implements Runnable interface.

The commonly used constructors of thread class are-

- Thread()
- Thread (String name)
- Thread(Runnable r)
- Thread(Runnable r,String name)

Commonly used methods of Thread class

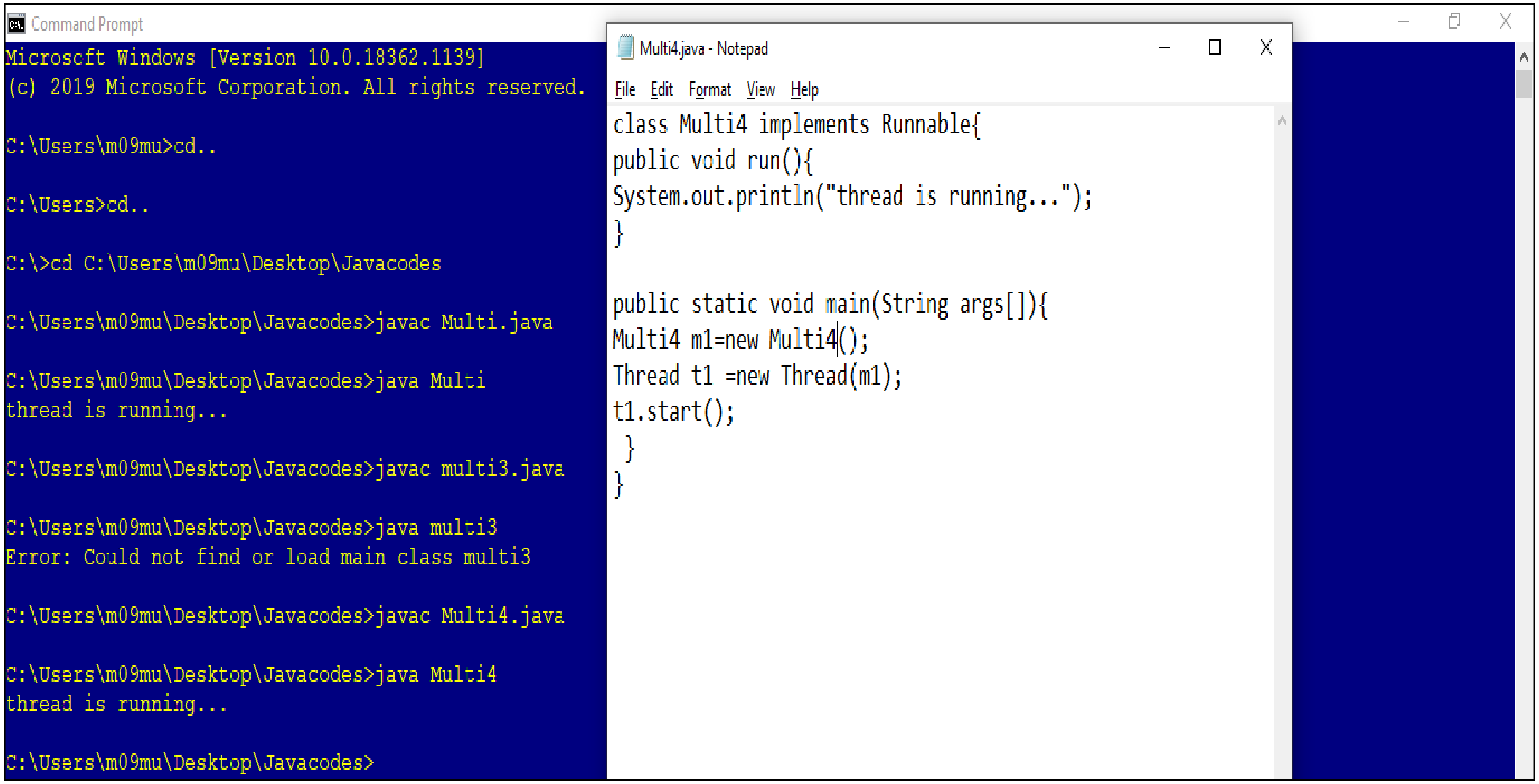
METHOD NAME	METHOD DESCRIPTION
public void run()	is used to perform action for a thread.
public void start()	starts the execution of the thread.JVM calls the run() method on the thread.
public void join()	waits for a thread to die.
public void setName(String name)	Changes the name of the thread
public Thread currentThread()	Returns the reference of currently executing thread
public int getId()	Returns the ID of the thread
public Thread.State getState()	Returns the state of the thread
public boolean isAlive()	Tests if the thread is alive
public void yield()	Pauses the currently executing thread temporarily
public void interrupt()	Interrupts the thread
public boolean isInterrupted()	Tests if a thread has been interrupted



```
Microsoft Windows [Version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\m09mu>cd..

C:\Users>cd..

C:\>cd C:\Users\m09mu\Desktop\Javacodes

C:\Users\m09mu\Desktop\Javacodes>javac Multi.java

C:\Users\m09mu\Desktop\Javacodes>java Multi
thread is running...

C:\Users\m09mu\Desktop\Javacodes>
```

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

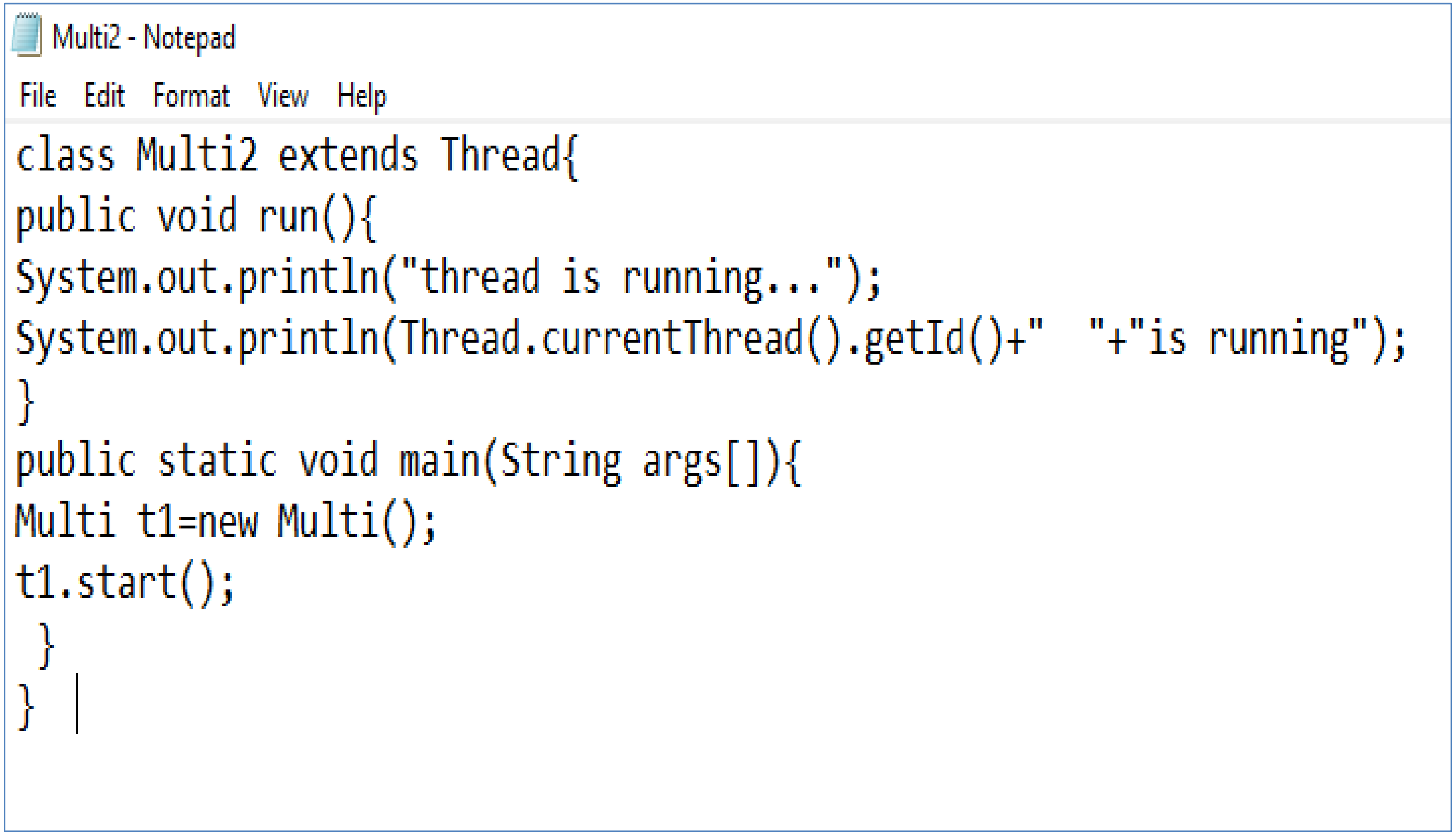
Java Thread Example by implementing Runnable interface



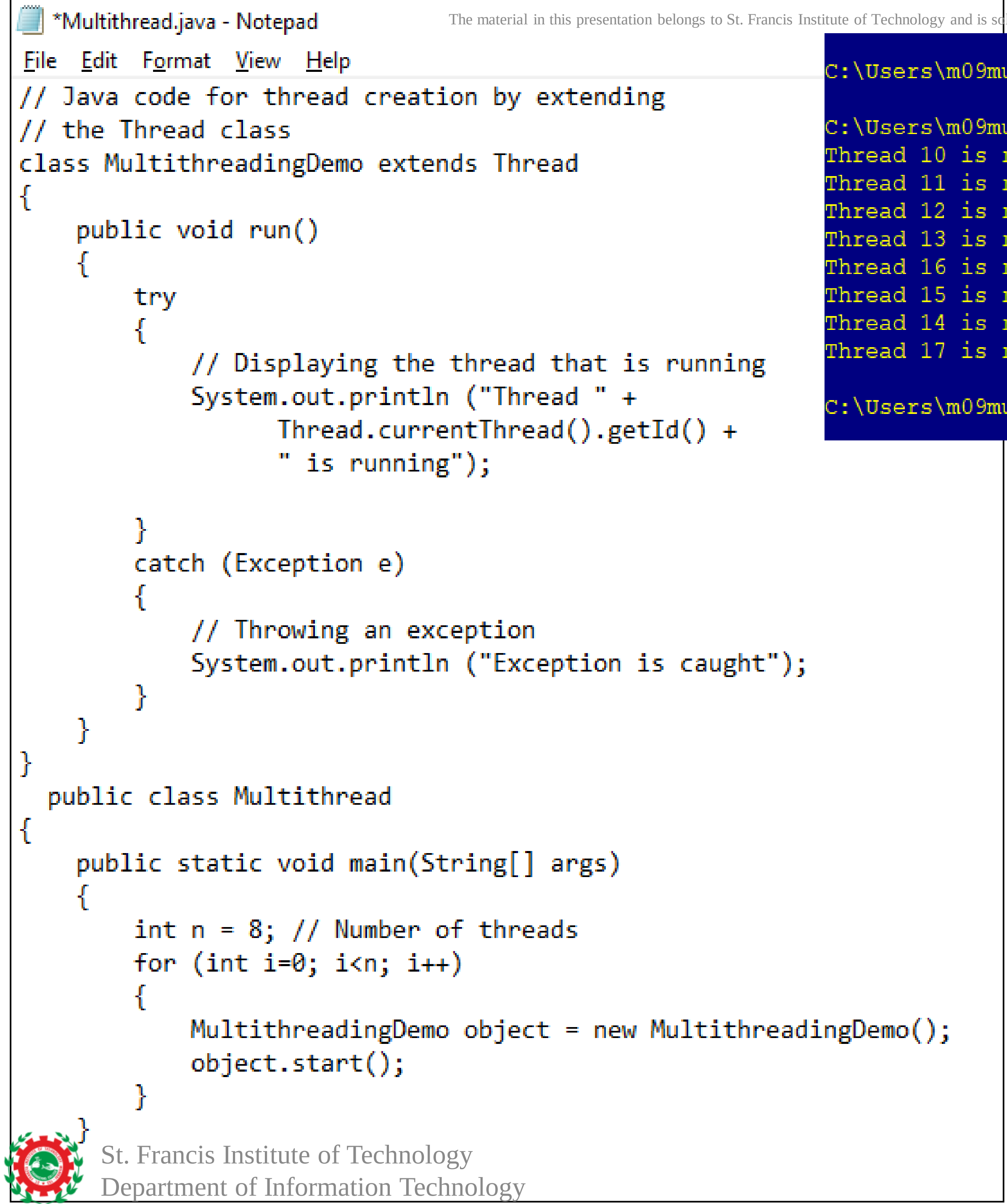
```
Microsoft Windows [Version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\m09mu>cd..

C:\Users>cd..

C:\>cd C:\Users\m09mu\Desktop\Javacodes

C:\Users\m09mu\Desktop\Javacodes>javac Multi.java

C:\Users\m09mu\Desktop\Javacodes>java Multi
thread is running...

C:\Users\m09mu\Desktop\Javacodes>javac multi3.java

C:\Users\m09mu\Desktop\Javacodes>java multi3
Error: Could not find or load main class multi3

C:\Users\m09mu\Desktop\Javacodes>javac Multi4.java

C:\Users\m09mu\Desktop\Javacodes>java Multi4
thread is running...

C:\Users\m09mu\Desktop\Javacodes>
```

```java
class Multi4 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi4 m1=new Multi4();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```

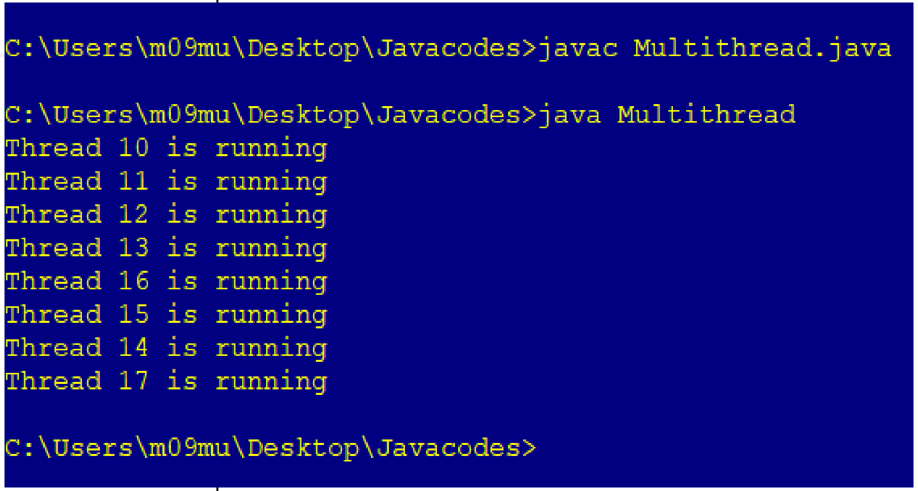
- If you are not extending the Thread class, your class object would not be treated as a thread object.
- So you need to explicitely create Thread class object.
- We are passing the object of your class that implements Runnable so that your class run() method may execute.



```
class Multi2 extends Thread{
public void run(){
System.out.println("thread is running...");
System.out.println(Thread.currentThread().getId()+"  "+"is running");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```



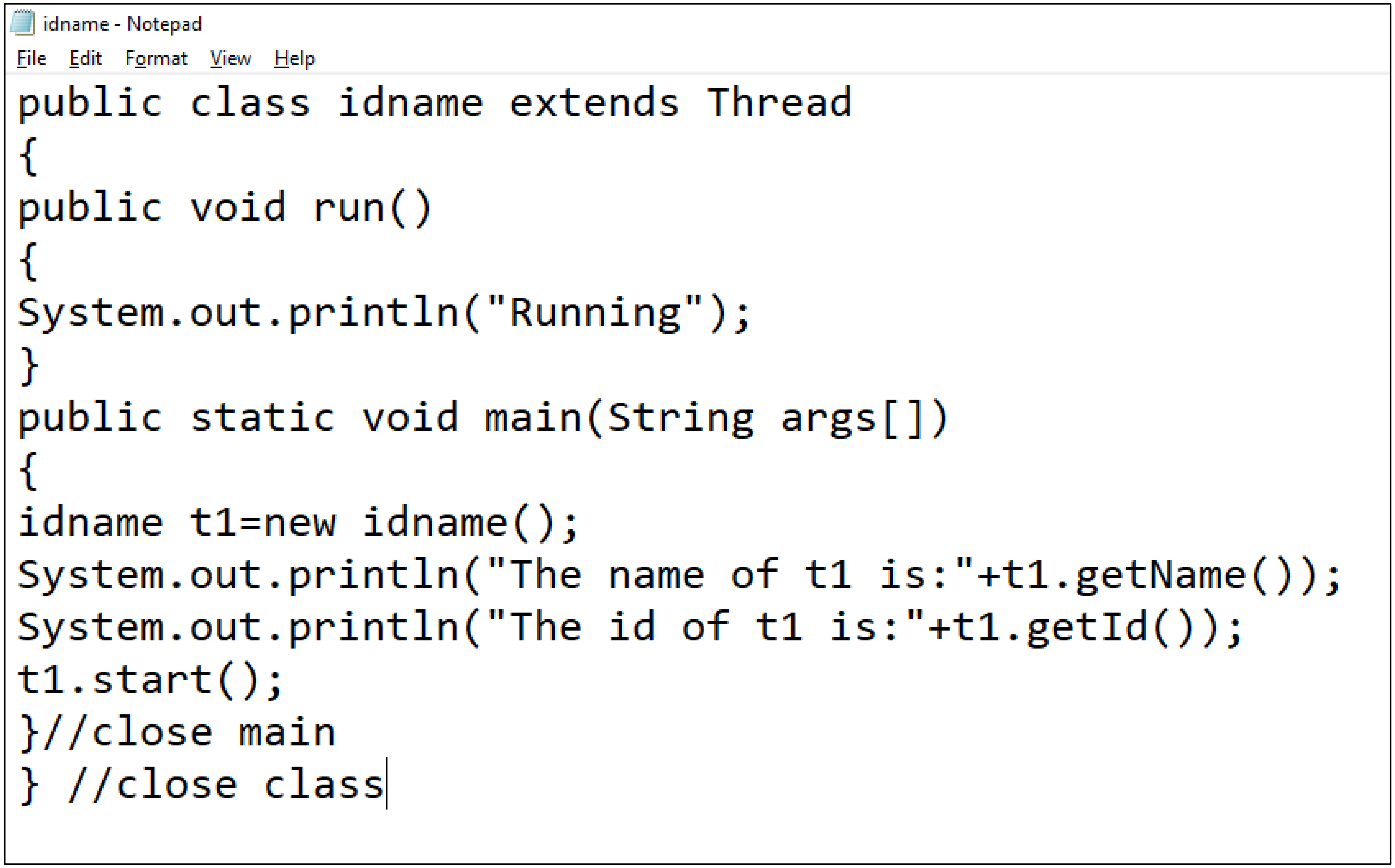
```
// Java code for thread creation by extending
// the Thread class
class MultithreadingDemo extends Thread
{
    public void run()
    {
        try
        {
            // Displaying the thread that is running
            System.out.println ("Thread " +
                  Thread.currentThread().getId() +
                  " is running");

        }
        catch (Exception e)
        {
            // Throwing an exception
            System.out.println ("Exception is caught");
        }
    }
}
  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();
        }
    }
```



```
C:\Users\m09mu\Desktop\Javacodes>javac Multithread.java

C:\Users\m09mu\Desktop\Javacodes>java Multithread
Thread 10 is running
Thread 11 is running
Thread 12 is running
Thread 13 is running
Thread 16 is running
Thread 15 is running
Thread 14 is running
Thread 17 is running

C:\Users\m09mu\Desktop\Javacodes>
```

Other way.....

```
public class Multithread1 extends Thread {
public void run() {
for (int i = 1; i <= 5; i++) {
System.out.println(" Thread name = "+ Thread.currentThread().getName());
}
}
public static void main(String[] args) {
Multithread1 t1 = new Multithread1();
t1.start();
Multithread1 t2 = new Multithread1();
t2.start();
Multithread1 t3 = new Multithread1();
t3.start();
}
}
```



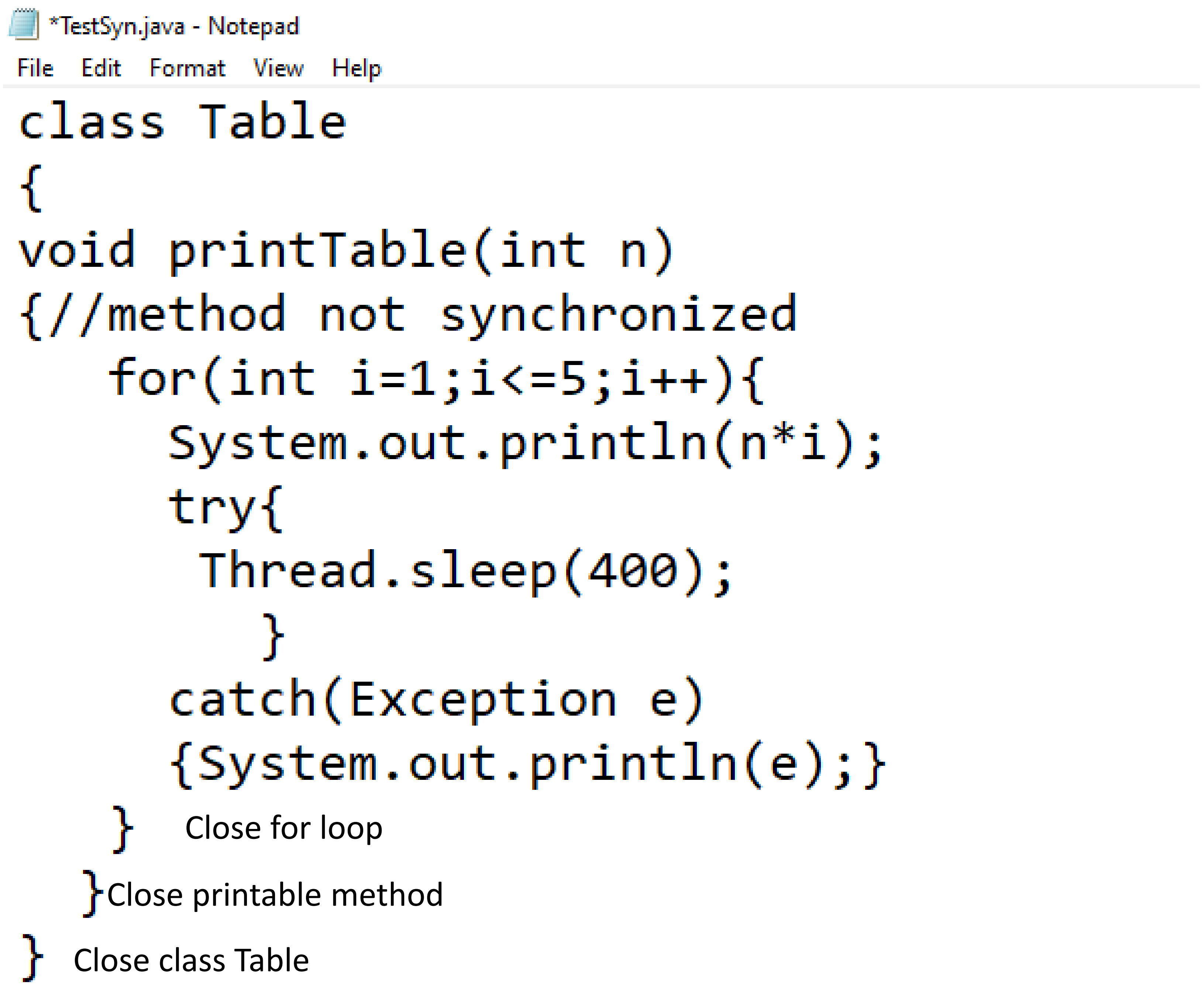
```
public class idname extends Thread
{
public void run()
{
System.out.println("Running");
}
public static void main(String args[])
{
idname t1=new idname();
System.out.println("The name of t1 is:"+t1.getName());
System.out.println("The id of t1 is:"+t1.getId());
t1.start();
}//close main
} //close class
```

- The **getId()** method is used to return the thread identifier. The thread ID is a unique positive number which was generated at the time of thread creation.
- The thread ID remains unchanged during its lifetime.

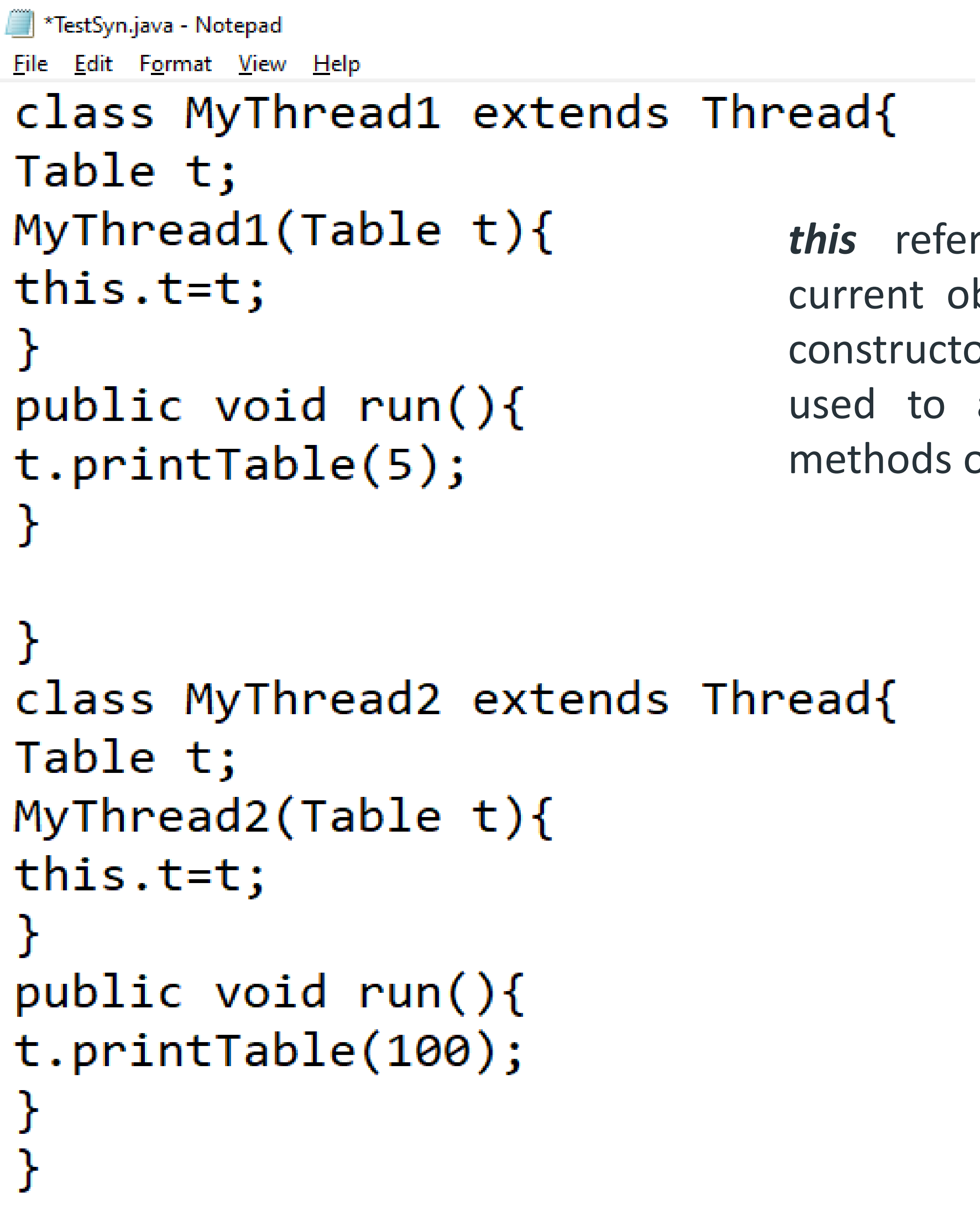
5.1: Implementing synchronization, Message passing

5.3: Communication and Synchronization

- Multithreaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results.
- So it needs to be made sure by some synchronization method that only one thread can access the resource at a given point of time.
- Java provides a way of creating threads and synchronizing their task by using synchronized blocks.
- Synchronized blocks in Java are marked with the synchronized keyword.
- A synchronized block in Java is synchronized on some object.
- All synchronized blocks synchronized on the same object can only have one thread executing inside them at a time.
- Synchronized method is used to lock an object for any shared resource.
- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.
- Show and explain the program (TestSyn.java)…..For non Synchronized working
- Show and explain the program (TestSyn2.java) [for synchronized working]



```
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);}
   }    Close for loop
  }Close printable method
}  Close class Table
```



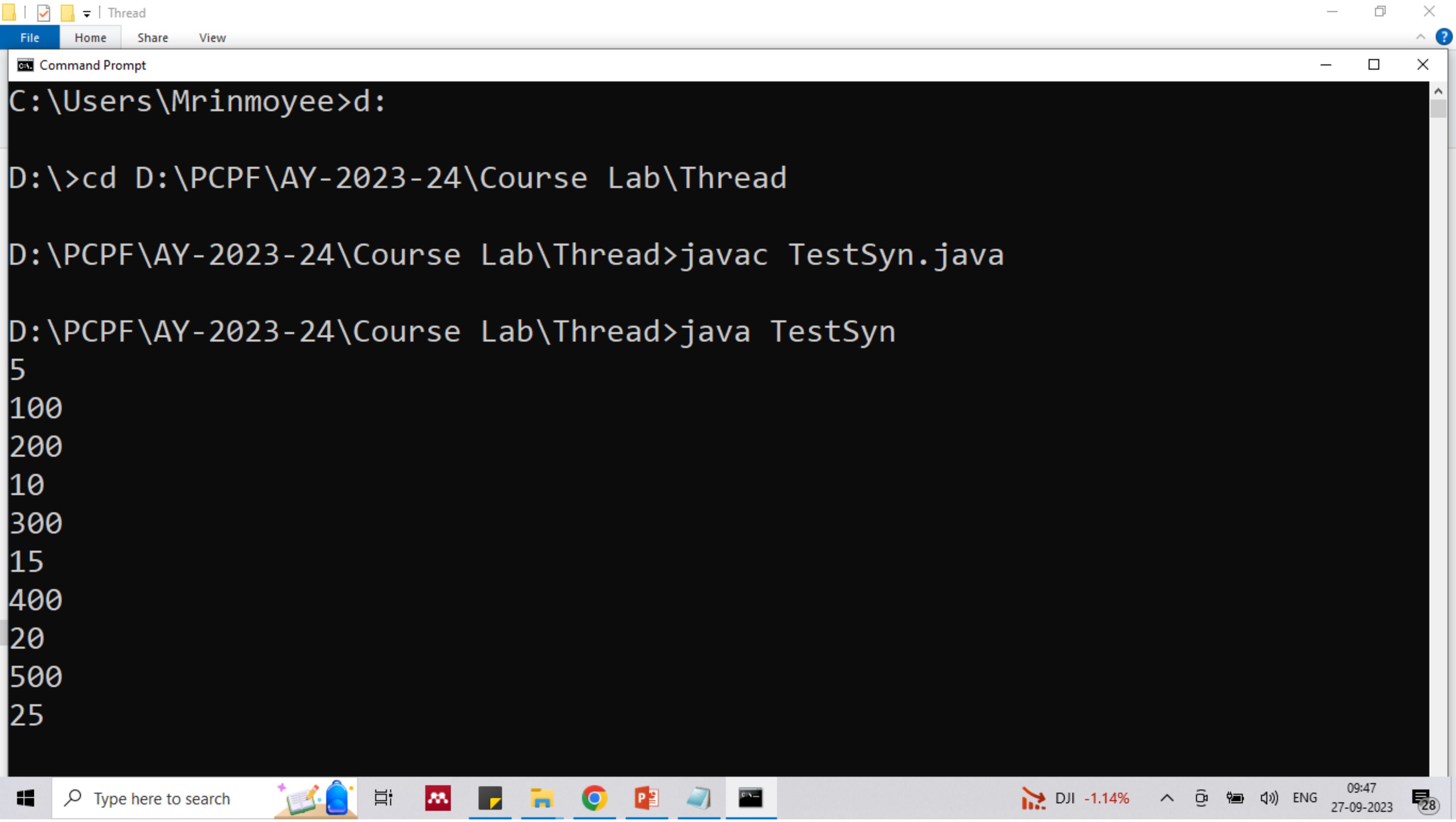
```
class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

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

this reference variable refers to the current object on which the method or constructor is being invoked. It can be used to access instance variables and methods of the current object.

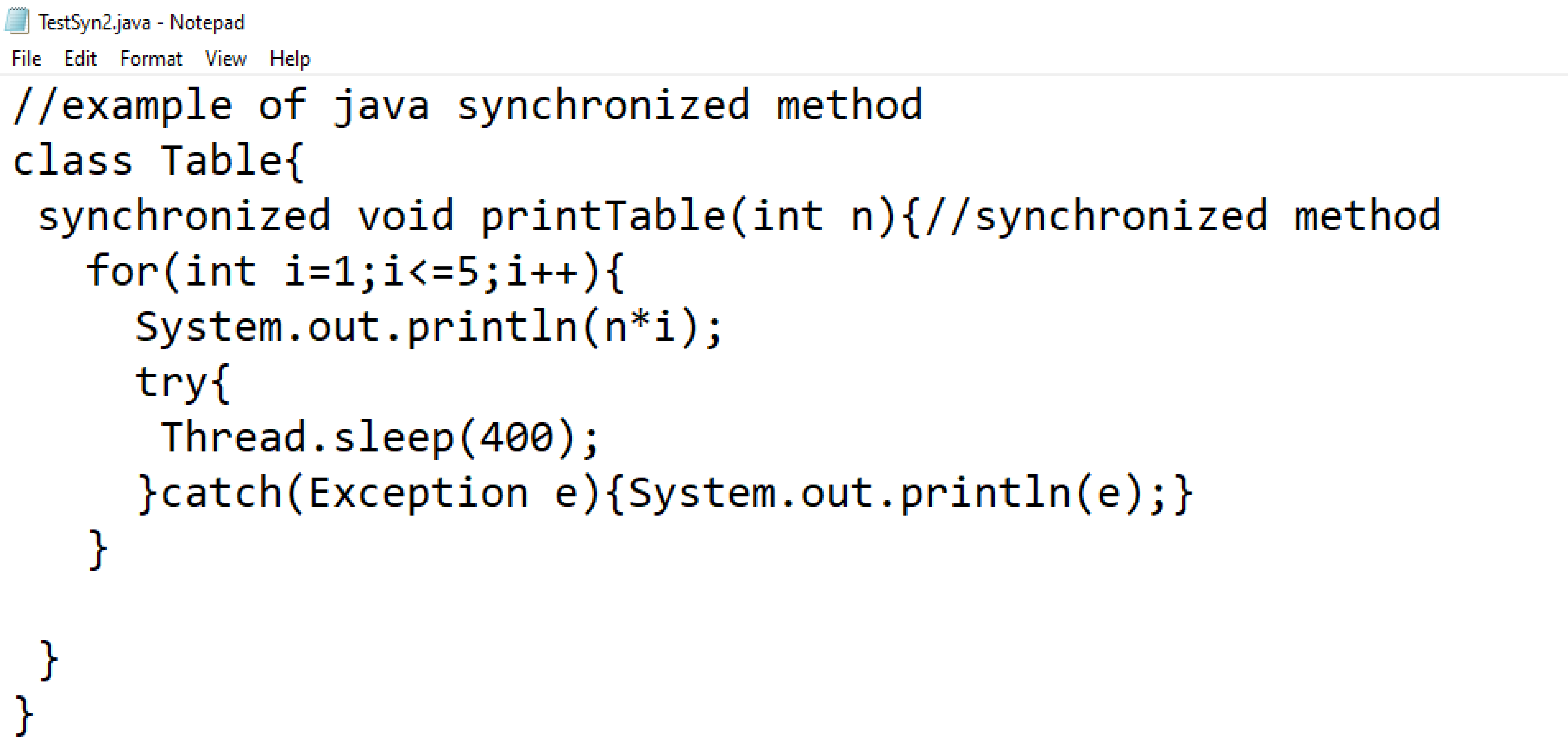
```
class TestSyn{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

When a new thread is created, the run() method is not called automatically. We must call the thread's start() method to start the thread's work.

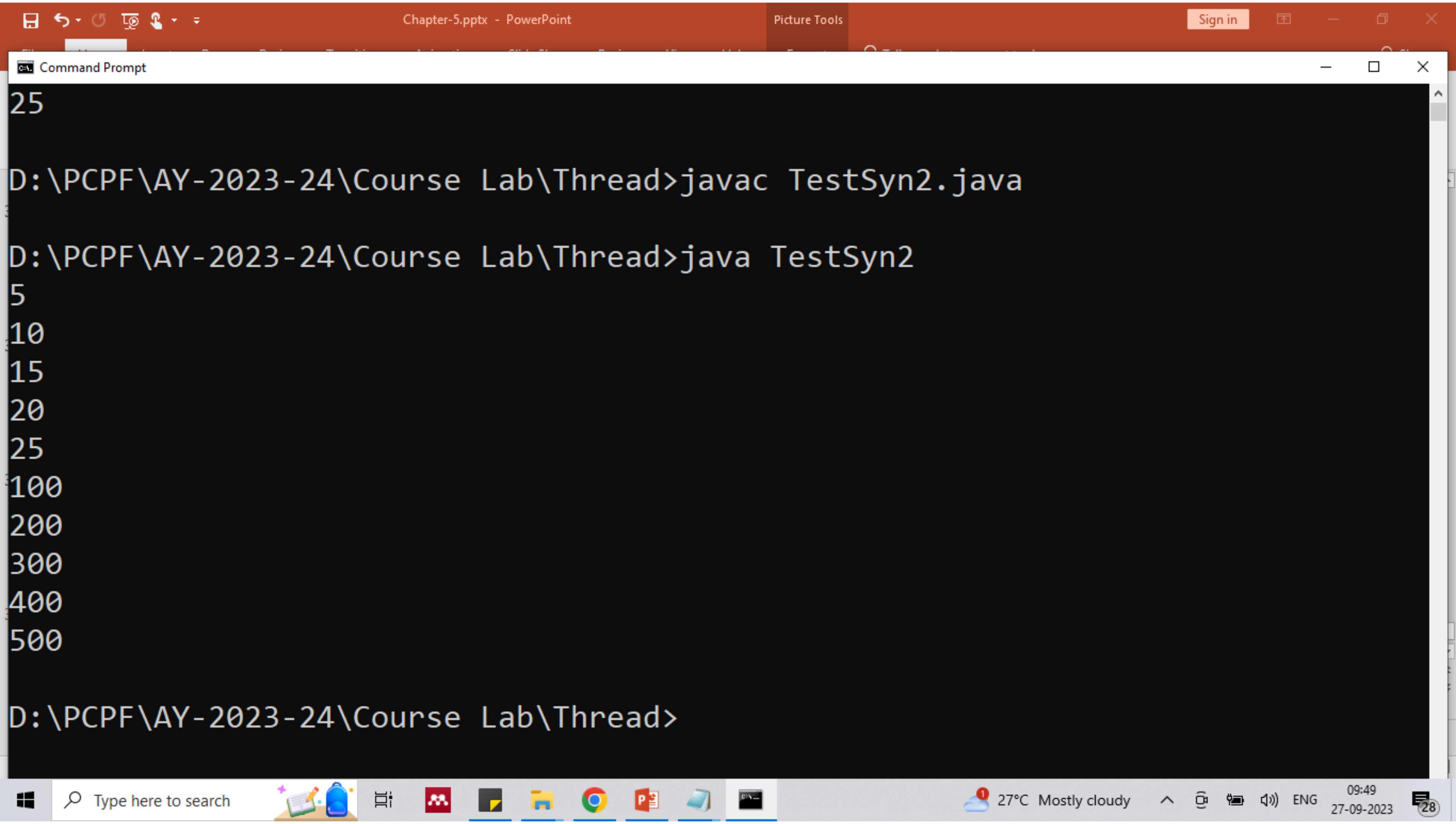
This method does not take any parameters.
This method does not return a value.
This method does not throw any exception.



```
C:\Users\Mrinmoyee>d:

D:\>cd D:\PCPF\AY-2023-24\Course Lab\Thread

D:\PCPF\AY-2023-24\Course Lab\Thread>javac TestSyn.java

D:\PCPF\AY-2023-24\Course Lab\Thread>java TestSyn
5
100
200
10
300
15
400
20
500
25
```



```
//example of java synchronized method
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}
```



```
25

D:\PCPF\AY-2023-24\Course Lab\Thread>javac TestSyn2.java

D:\PCPF\AY-2023-24\Course Lab\Thread>java TestSyn2
5
10
15
20
25
100
200
300
400
500

D:\PCPF\AY-2023-24\Course Lab\Thread>
```

Issues in Multithreaded Applications

- Common Issues in multithreaded Applications
- For all the advantages of using multiple threads, they add complexity and can create tough bugs to solve. There are some common scenarios where you may encounter challenges with debugging multithreaded applications. These include:
  - Investigating data access issues where two threads are reading and modifying the same data. Without the proper use of locking mechanisms, data inconsistency and dead-lock situations can arise.
  - Thread starvation and resource contention issues arise if many threads are trying to access a shared resource.
  - Display issues can surface if threads are not coordinated correctly when displaying data.

What Is Multithreading Used For?

- The main reason for incorporating threads into an application is to improve its performance.
- Performance can be expressed in multiple ways:
  - A web server will utilize multiple threads to simultaneous process requests for data at the same time.
  - An image analysis algorithm will launch multiple threads at a time and segment an image into quadrants to apply filtering to the image.
  - A ray-tracing application will launch multiple threads to compute the visual effects while the main GUI thread draws the final results.
- Multithreading also leads to minimization and more efficient use of computing resources.
  - Application responsiveness is improved as requests from one thread do not block requests from other threads.
- Multithreading is less resource-intensive than running multiple processes at the same time.
  - There is much more overhead, time consumption, and management involved in creating processes as compared to creating and managing threads.