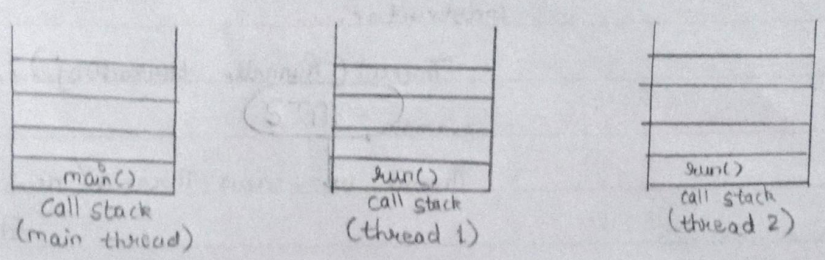
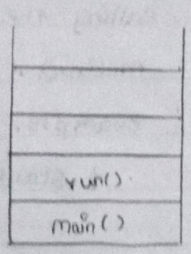
Practical No. 14

Aim : Create, debug and run java programs based on threads by extending thread class.

Diagram :



run() invoked by start() method



run() invoked directly from main method.

PAGE NO.
DATE

Practical No. 13 14

Aim : Create, debug and run Java programs based on threads by extending Thread class.

Theory :

What is Thread class?

- java.lang.Thread is an class that is to be extended by a class which whose objects are to be treated as threads.
- Java provides a thread class that has various method calls in order to manage the behaviour of the threads by providing constructors and methods to perform operations on threads.
- Thread class is used in cases where there is ~~no need of~~ no need of extending any other class than the thread class.

What are the steps to create a class by extending thread class?

→ Step 1:

You need to override the run() method available in the Thread class. This method, i.e run() provides ~~an~~ the entry point for the thread and the code to be executed by thread should go here.

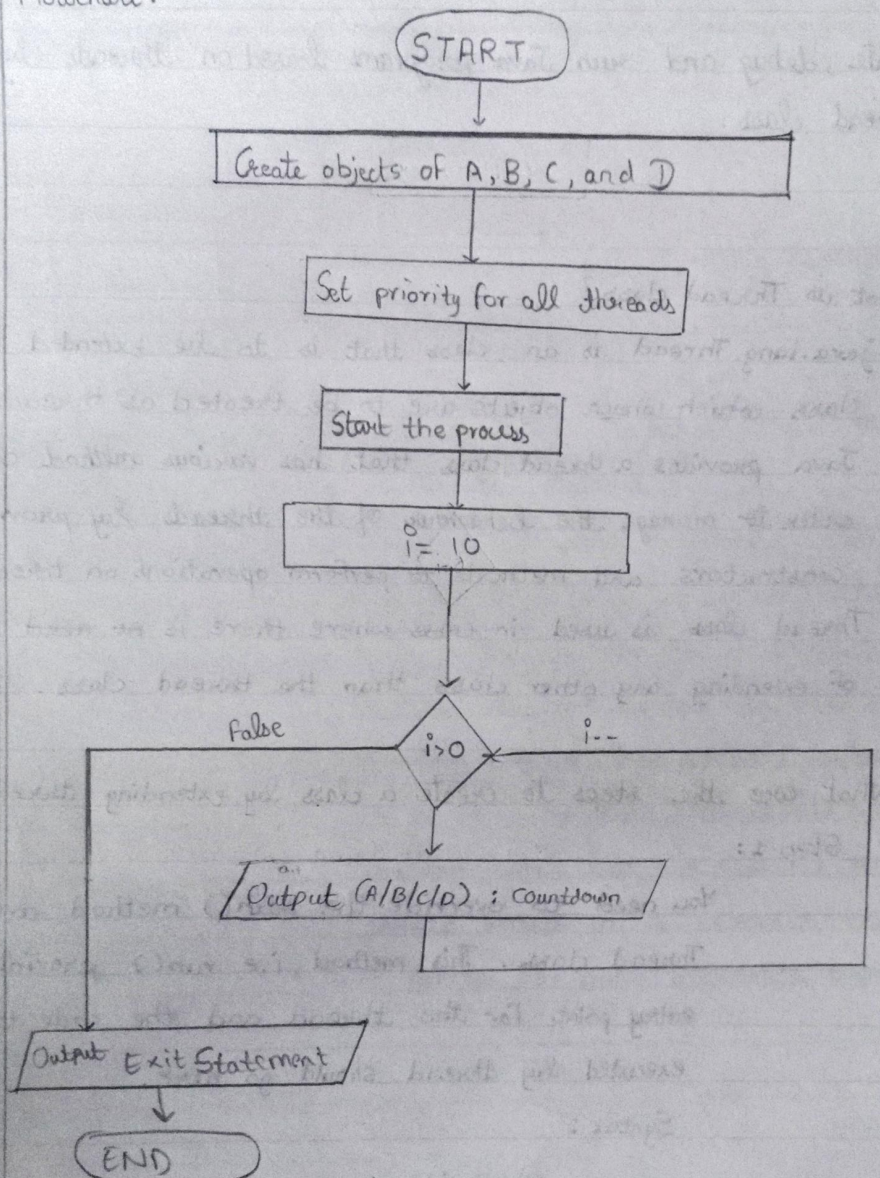
Syntax :

```
@override
public void run() {
    // Line of code(s).
}
```

Step 2:

Once the object of thread class is created, you can start it by calling the start() ~~method()~~ method.

Flowchart:



Conclusion:

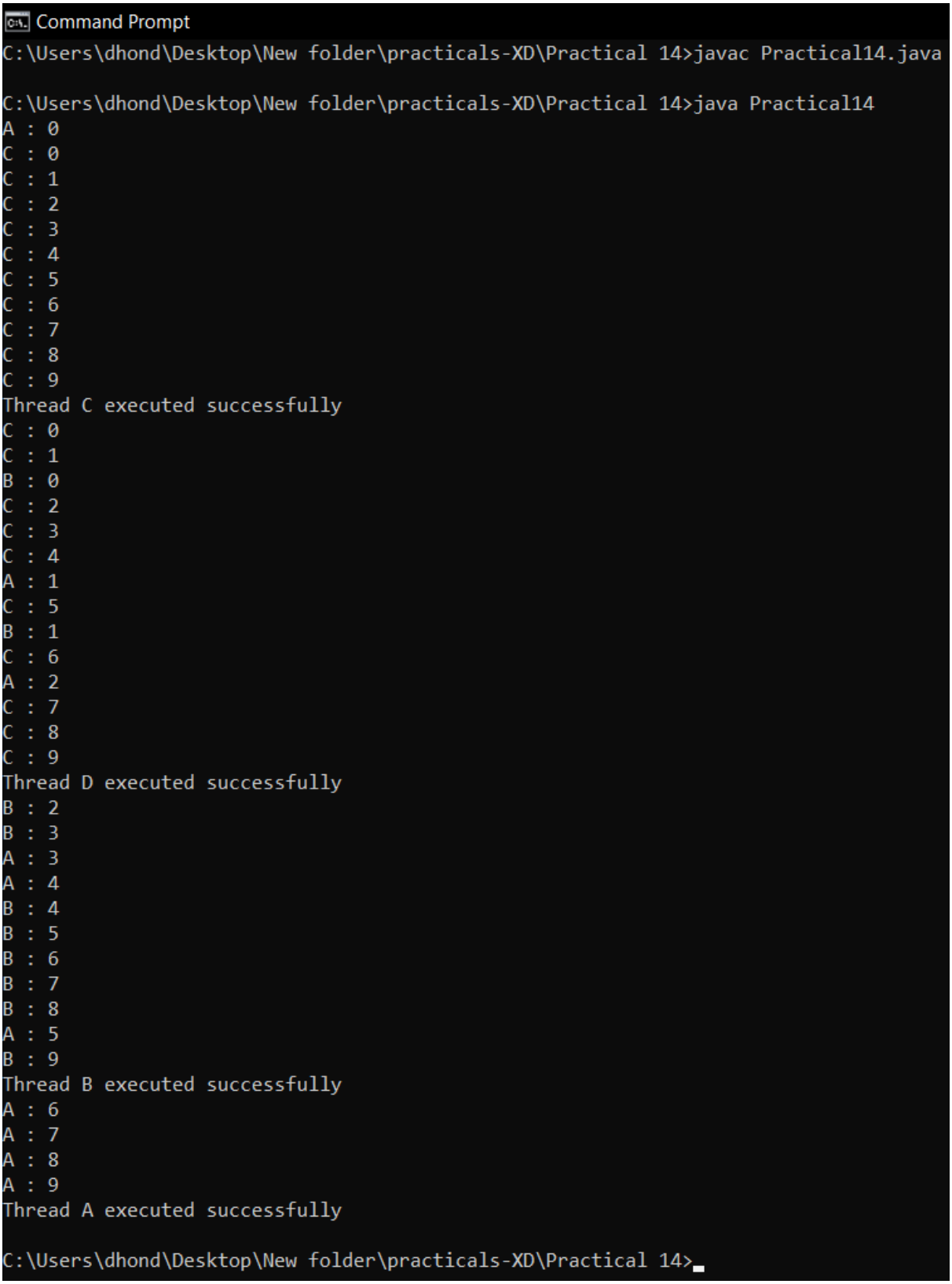
Hence, by performing this practical I got to learn about how to create java programs which can perform multithreading by extending thread classes. I also created, debugged and executed java programs based on the concept of threads by extending classes.

Code:

```
class A extends Thread{
    synchronized public void run(){
        for(int i = 0; i <10; i++){
            System.out.println("A : " + i);
            try {

            }catch (Exception e){}
        }
        System.out.println("Thread A executed successfully");
    }
}

class B extends Thread{
    synchronized public void run(){
        for(int i = 0; i <10; i++){
            System.out.println("B : " + i);
            try {

            }catch (Exception e){}
        }
        System.out.println("Thread B executed successfully");
    }
}

class C extends Thread{
    synchronized public void run(){
        for(int i = 0; i <10; i++){
            System.out.println("C : " + i);
            try {

            }catch (Exception e){}
        }
        System.out.println("Thread C executed successfully");
    }
}

class D extends Thread{
    synchronized public void run(){
        for(int i = 0; i <10; i++){
            System.out.println("C : " + i);
            try {

            }catch (Exception e){}
        }
        System.out.println("Thread D executed successfully");
    }
}
```

```java
class Practical14{

    public static void main(String[] args) {

        A a = new A();
        B b = new B();
        C c = new C();
        D d = new D();

        a.setPriority(Thread.MIN_PRIORITY);
        b.setPriority(Thread.NORM_PRIORITY);
        c.setPriority(Thread.MAX_PRIORITY);
        d.setPriority(Thread.MIN_PRIORITY);

        a.start();
        b.start();
        c.start();
        d.start();



    }

}
```

Output:


Command Prompt
C:\Users\dhond\Desktop\New folder\practicals-XD\Practical 14>javac Practical14.java
C:\Users\dhond\Desktop\New folder\practicals-XD\Practical 14>java Practical14
A : 0
C : 0
C : 1
C : 2
C : 3
C : 4
C : 5
C : 6
C : 7
C : 8
C : 9
Thread C executed successfully
C : 0
C : 1
B : 0
C : 2
C : 3
C : 4
A : 1
C : 5
B : 1
C : 6
A : 2
C : 7
C : 8
C : 9
Thread D executed successfully
B : 2
B : 3
A : 3
A : 4
B : 4
B : 5
B : 6
B : 7
B : 8
A : 5
B : 9
Thread B executed successfully
A : 6
A : 7
A : 8
A : 9
Thread A executed successfully
C:\Users\dhond\Desktop\New folder\practicals-XD\Practical 14>

PAGE NO.
DATE

, which executes a call to run method.

Syntax:

```
ClassThread t = new Thread( ClassThread( );
    t.start();
```

Why is it necessary to start() the thread object and not to use the run() method?

- If we try to use 'ThreadObj.run();' instead of using start(), then we would notice that the program works similar to a single threaded program.
- This is because when we call the run() method directly from the main() method, the thread starts in the current call stack.
- Instead, when we use start() method in main(), the thread starts in a seperate call stack.
- The stacks for both scenarios are shown in Figure 14.1.

Conclusion:

Hence, by performing this practical I got to learn about how to create java programs which can perform multithreading by extending thread classes. I also created, debugged and executed java programs based on the concept of threads by extending classes.