Name : Pratyay Dhond

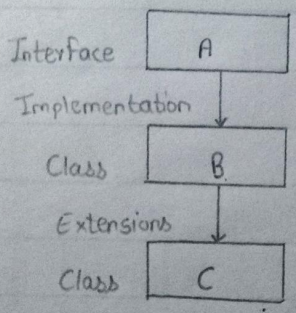
Department : IT

Roll No : 1907011

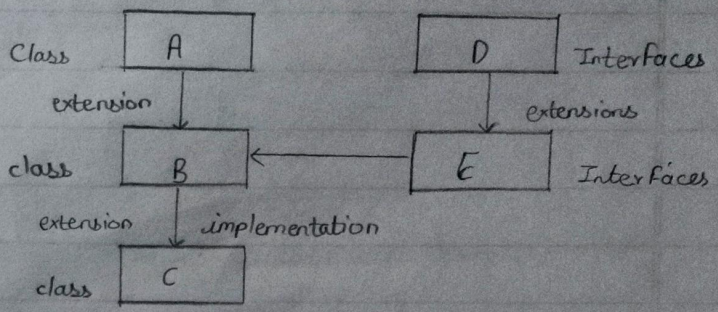
Practical No. 09

Aim: Create, debug and run java programs based on interfaces.

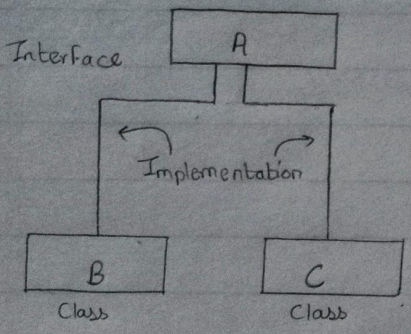
Diagram:



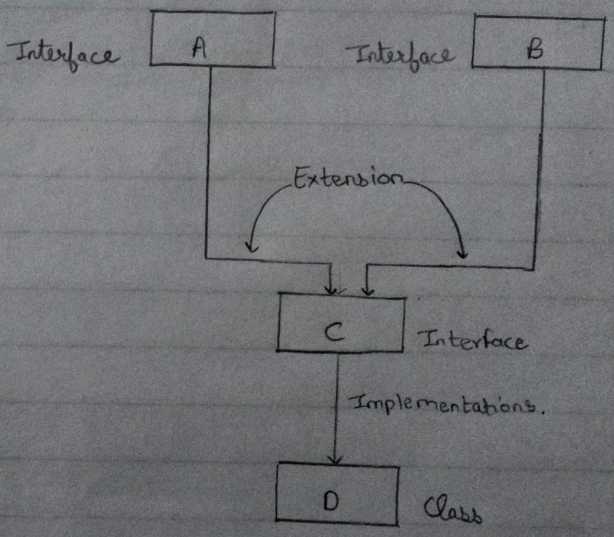
(a)



(b)



(c)



(d)

Practical No. 9

Aim : Create, debug and run java programs based on interfaces.

Theory :

What is an interface?

→ • An interface is almost like a class with a major difference being that interfaces define only abstract methods and final fields.

- Syntax:

```
interface InterfaceName {
    variable declaration
    Method declaration.
}
```

here, 'interface' is the Keyword, and InterfaceName is any valid Java variable (just like class names).

- Variables are declared as follows:

```
static final type VariableName = Value;
```

example:

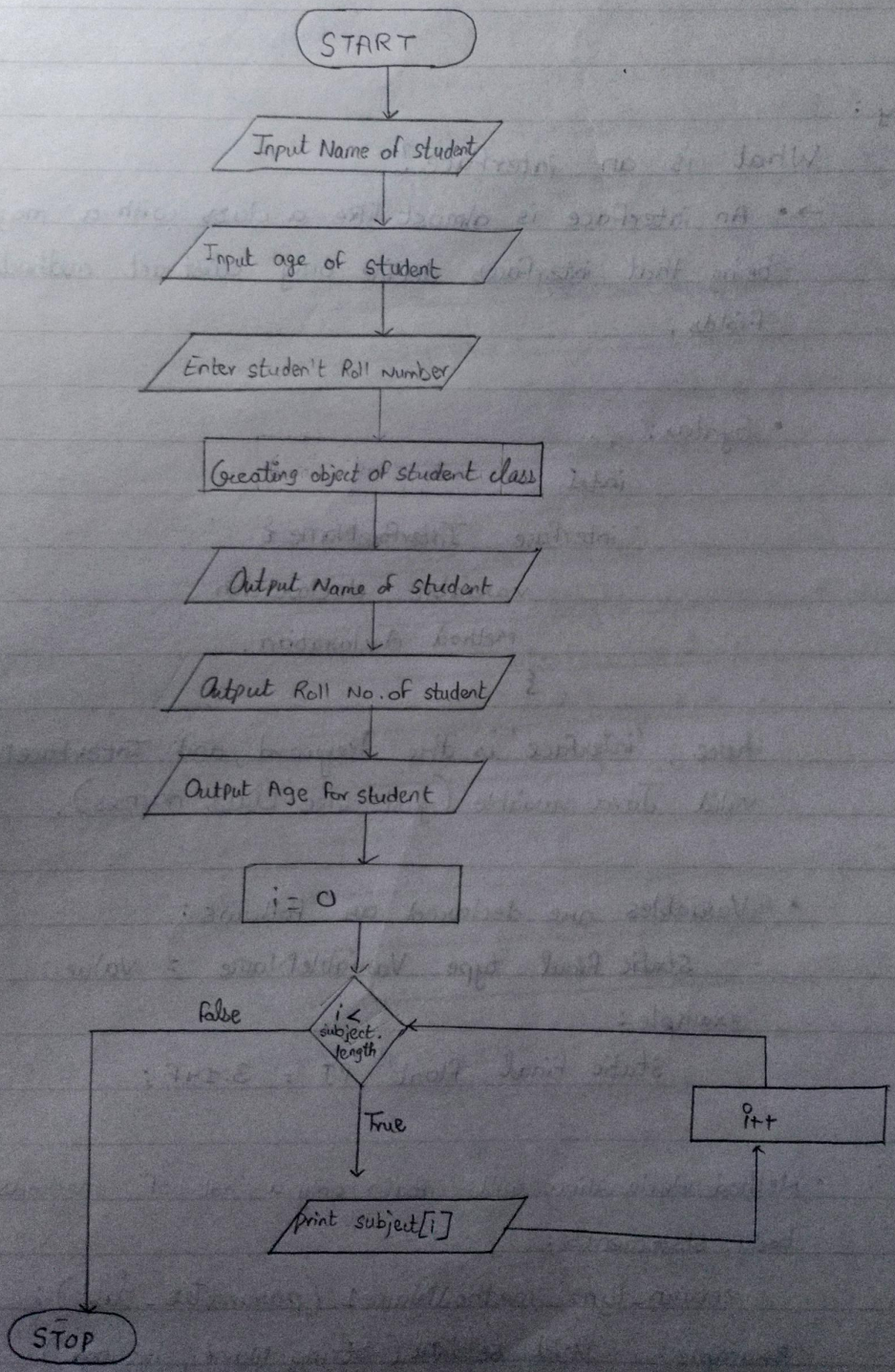
```
Static final float PI = 3.14F;
```

- Method declaration will contain only a list of methods without any body statements:

```
return-type methodName1 (parameter-list);
```

example: `void setData (String Name, int age);`

Flowchart :


START
Input Name of student
Input age of student
Enter student Roll Number
Creating object of student class
Output Name of student
Output Roll No. of student
Output Age for student
i = 0
i < subject.length
False
True
print subject[i]
i++
STOP

Extending Interfaces:

- An interface can be sub-interfaced from other interfaces.
- The new sub interface will inherit all the members of the sub interface in the manner similar to subclasses.
- This is achieved using the keyword 'extends'.
- Syntax:

```
interface name1 extends name2 {
  // body of name 1
}
```

- We can also combine several interfaces together into a single interface. Syntax:

```
interface a {
  // body of a
}

interface b {
  // body of b
}

interface c extends a, b {
  // body of c
}
```

- Interfaces cannot extend classes as it would violate the rule of interfaces having only final variables and abstract methods.

Conclusion :

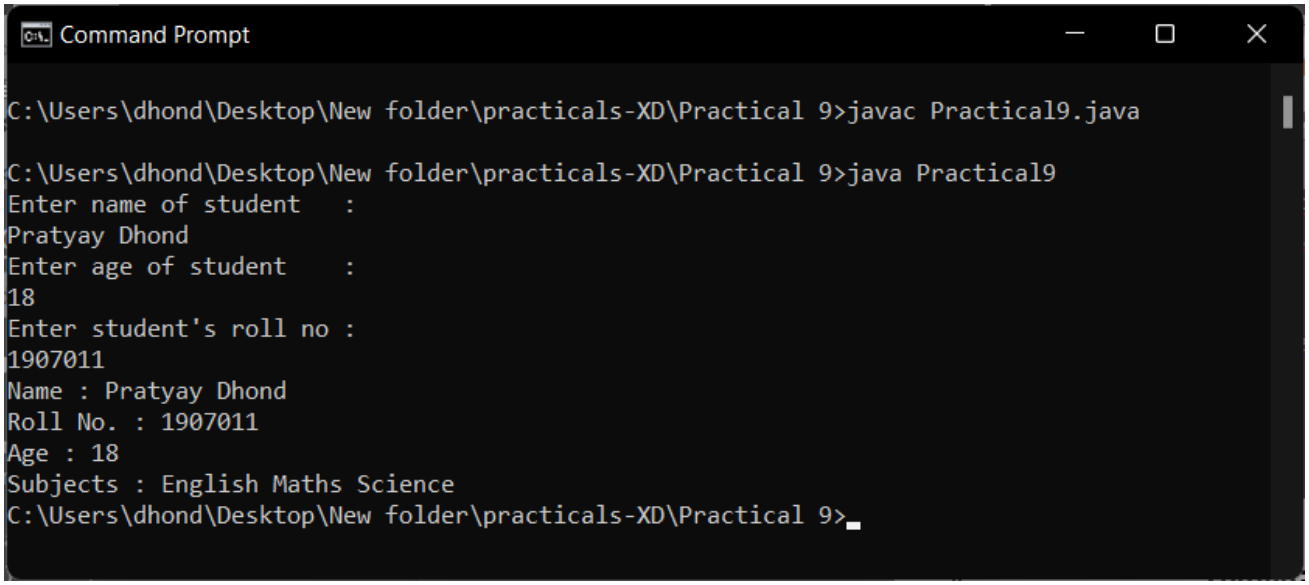
Hence, by performing this practical, I learnt about the concept of interfaces and their implementation. I also created, developed, debugged and executed java programs on interfaces

Code :

```
import java.util.Scanner;

interface Student{
    final String subjects[] = {"English","Maths","Science"};
    void display();
    void showName();
    void showRollNo();
    void showAge();
}

class StudentInfo implements Student{
    String name;
    int rollNo;
    int age;

    StudentInfo(String name, int age, int rollNo){
        this.name = name;
        this.age = age;
        this.rollNo = rollNo;

    }

    public void showName() {
        System.out.println("Name : " + name);
    }

    public void display(){
        showName();
        showRollNo();
        showAge();
        System.out.print("Subjects : ");
        for(String a : subjects){
            System.out.print(a + " ");
        }
    }

    public void showRollNo(){
        System.out.println("Roll No. : " + rollNo);
    }

    public void showAge(){
        System.out.println("Age : "  + age);
    }
}
```

```java
class Practical9{

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String name = " ";
        int age = 0;
        int rollNo = 0;
        System.out.println("Enter name of student   : ");
       name = sc.nextLine();
        System.out.println("Enter age of student    : ");
        age = sc.nextInt();
        System.out.println("Enter student's roll no : ");
        rollNo = sc.nextInt();

        StudentInfo student1 = new StudentInfo(name,age,rollNo);
        student1.display();
    }

}
```

Output:



Implementing interfaces :

- Interfaces are used as a "superclass" whose properties are inherited by classes.
- Syntax :

```
class className { implement InterfaceName{
    // body of className
}
```

- A more general / in use form of 'implements' may look like :

```
class className implements extends Superclass
    implements interface1, interface2 .... {
    // body of className
}
```

Thus, a class can extend ~~of~~ another class while implementing interface.

~~Code:~~

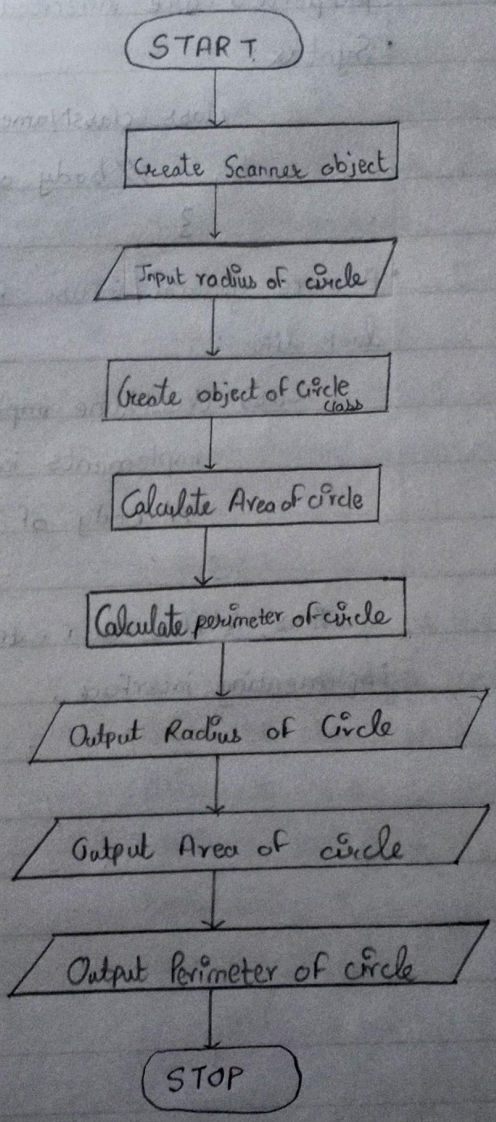
Conclusion :

Hence, by performing this practical, I learnt about the concept of interfaces and their implementation. I also created, developed, debugged and executed java programs on interfaces.

Practical No. 10

Aim : Create, debug and run java programs based on package

Flowchart :



Practical No. 10

Aim: Create debug and run java programs based on Package.

Theory:

What is a package?

→ "Packages are Java language's way of grouping a variety of classes and/or interfaces together. Java packages act as "containers" for classes."

Java packages are further classified into two types:

i) Java API package
ii) User defined packages

- Creating Packages:

1. Declare package at the beginning of file using the form:
   package packageName;
2. Define the class that is to be put in the package and declare it public.
3. Create a subdirectory under the directory where the main source files are stored.
4. Store the listings as the classname.java file in the subdirectory created.
5. Compile the file. This creates .class file in the subdirectory.

- Accessing package:

1. The import statement is used when there are many references to a particular package or the package name is too long and unwieldy.

2. Syntax:

import package1 [.package2] [.package3] . className ;

OR

import package1 [.package2] [.package3] . * ;

- Using package :
  - Syntax :

import package1 . className ;

[This will import the specific classes from the package].

OR

import package1 . * ;

[This will import the all the classes from accessible classes from the package].

- What is Static import ?

→ • The feature of static import eliminates the need of qualifying a static member with the class name.

- Syntax :

import static package-name . Subpackage-name . class-name . Static-member ;

OR

import static package-name . Subpackage-name . class-name . * ;

Conclusion :

Hence, I learnt about the concepts of package and static import. I also coded, developed, debugged and executed programs based on the concept of packages.

Code:

Driver Class:

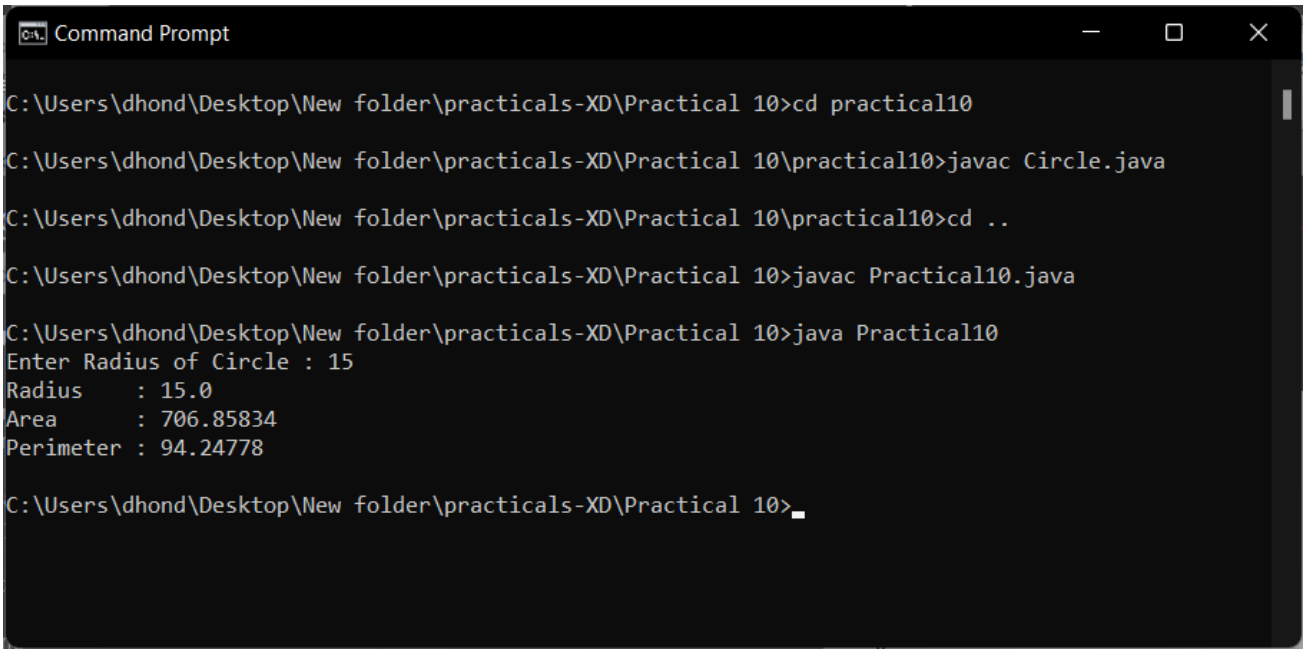
```
import practical10.Circle;
import java.util.Scanner;

class Practical10{
  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    float radius;
    System.out.print("Enter Radius of Circle : " );
    radius = sc.nextFloat();
    Circle circle = new Circle(radius);
    System.out.println("Radius    : " + circle.getRadius());
    System.out.println("Area      : " + circle.getArea());
    System.out.println("Perimeter : " + circle.getPerimeter());
  }
}
```

Class in subdirectory:

```
package practical10;

import static java.lang.Math.PI;

public class Circle{
  public float radius;
    public float _area;
    public float _perimeter;

    public Circle(float radius){
        this.radius = radius;
        area();
    perimeter();
    }

    public void area(){
        _area = (float) PI * radius * radius;
    }

    public void perimeter(){
        _perimeter = 2 * (float) PI * radius;
    }

    public float getRadius(){
        return radius;
    }
```

```
    public float getArea(){
      return _area;
    }

    public float getPerimeter(){
        return _perimeter;
    }

}
```

Output:



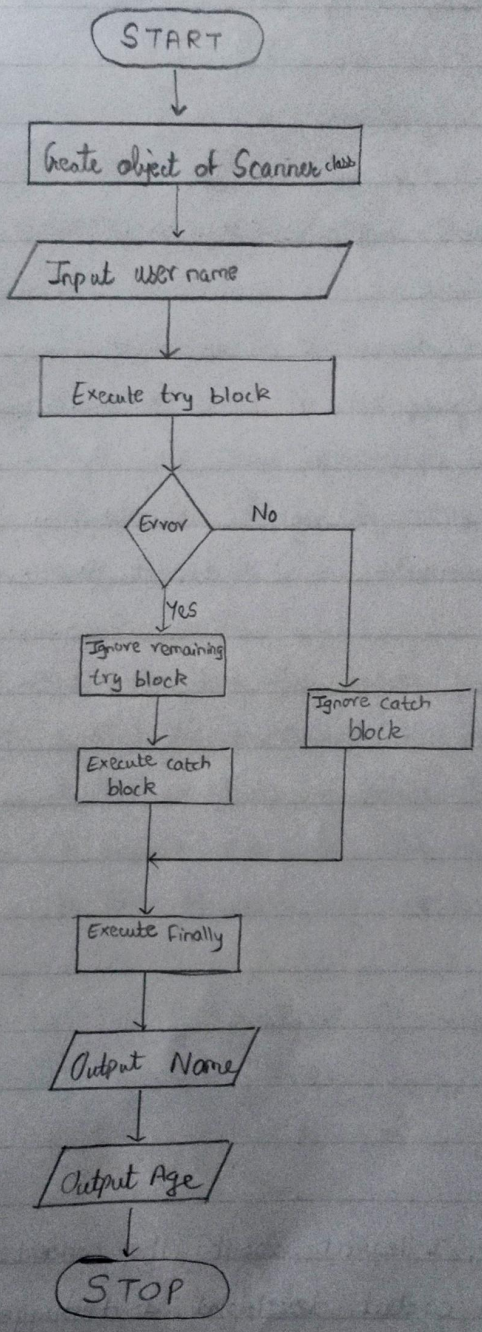
Conclusion :

Hence, I learnt about the concepts of package and static import. I also coded, developed, debugged and executed programs based on the concept of packages.

Practical No. 11

Aim : Create, debug and run java programs based on exception handling.

Flowchart :



Practical No. 11

Aim : Create, debug and ~~execute~~ run java programs based on exception handling.

Theory :

What is exception handling?

→ Exception handling is one of the powerful mechanism to handle ~~run~~ the runtime errors so that the normal flow of the application can be maintained.

- An exception is a ~~conditi~~ condition that is caused by a run-time error in the program.
- When the Java interpretor encounters an error such as dividing an integer by zero, it creates an Exception object and throws it (i.e. informs us that an error has occurred).

- What tasks does the error handling code perform :

→ 1. Find the problem (Hit the exception)
2. Inform that an error has occured (Throw the exception).
3. Recieve the error information (Catch the exception).
4. Take ~~th~~ corrective actions (Handle the exception).

- What are the types of exception?

→ i) Checked Exception
ii) Unchecked Exception

1) Checked Exception :

These exceptions are explicitly handled in the code itself with the help of try-catch

block. Checked exceptions are extended from the java.lang.Exception class.

2) Unchecked Exceptions :

These exceptions are not essentially handled in the program code; instead the JVM handles such exceptions.

- Unchecked exceptions are extended from the java.lang.RunTimeException.

- Syntax :

```
try {

    statement(s);

} catch(Exception e){

    statement(s);

} ~~catch~~
finally {
        Statement(s);
}
```

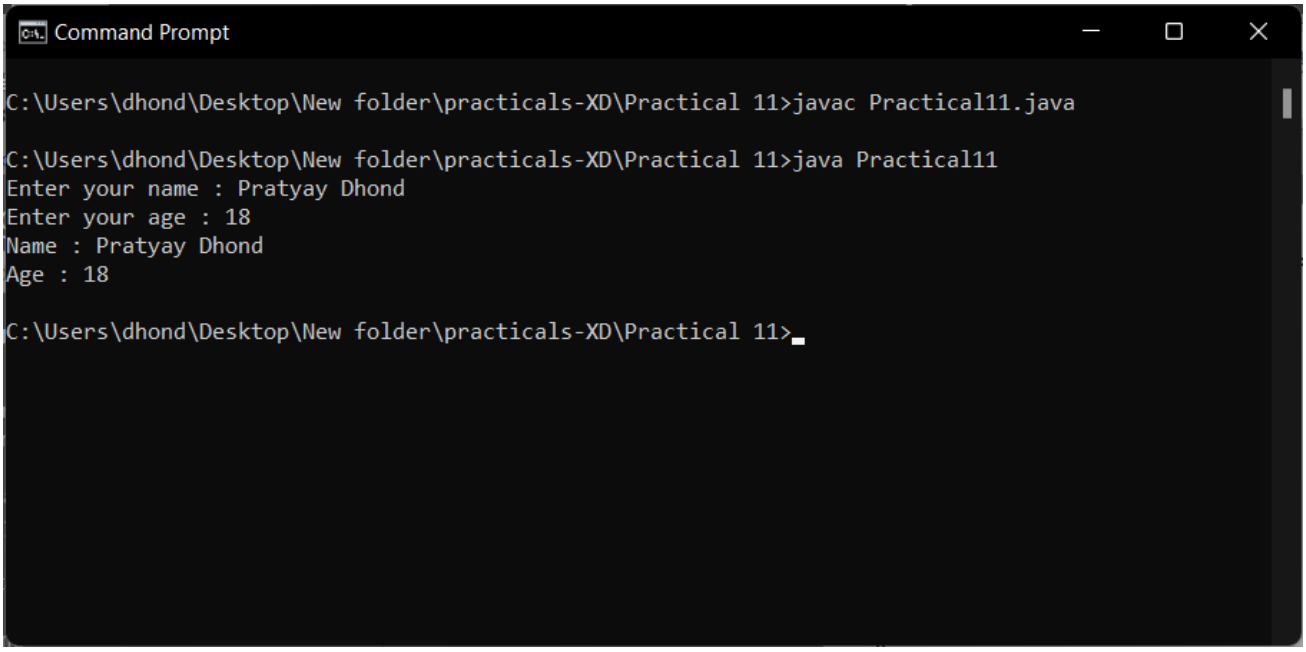
Conclusion:

Hence, I learnt about the concept of exception handling. I also created, debugged and executed java programs based on exception handling.

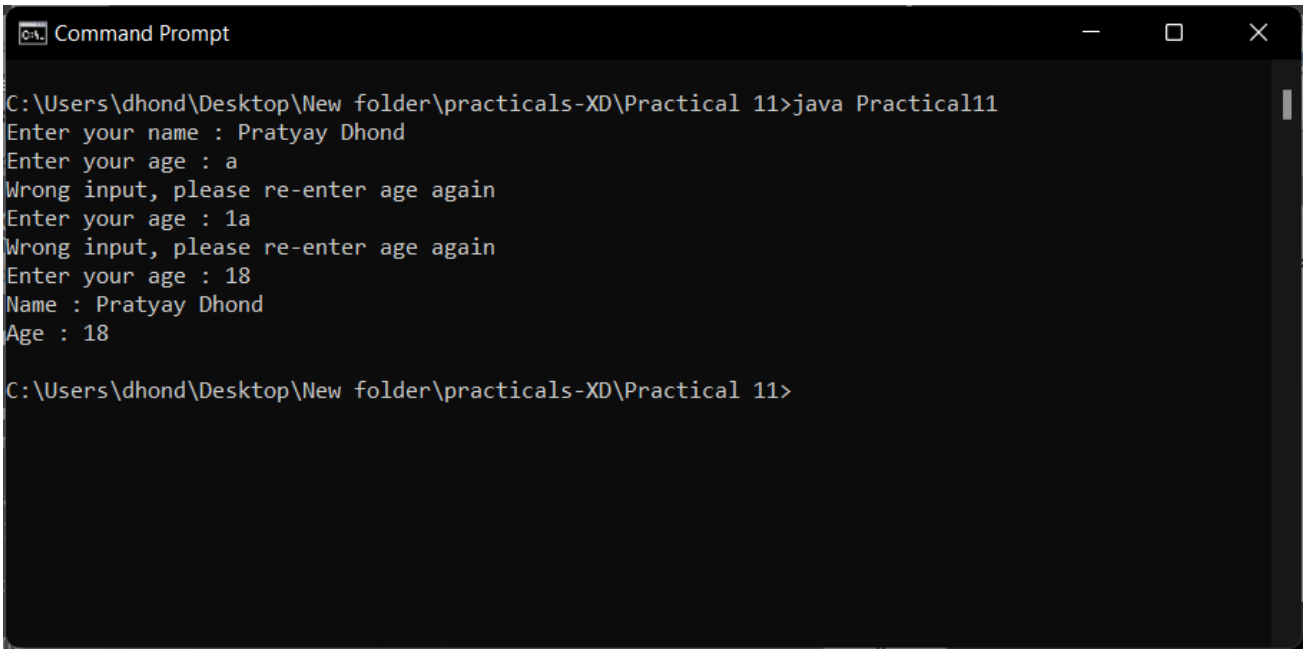
Code:

```
import java.util.InputMismatchException;
import java.util.Scanner;

class Practical11{

    public static int setAge(){
        Scanner sc = new Scanner(System.in);
        int age = 0;

        try {
            System.out.print("Enter your age : ");
            age = sc.nextInt();
        }catch(InputMismatchException e){
            System.out.println("Wrong input, please re-enter age again");
            age = setAge();
        }catch (Exception e){
            System.out.println("Unexpected error : ");
            System.out.println(e);
            System.out.println("Setting age to 0.");
        }finally {
           return age;
        }

    }

    public static void main(String[] args) {
        int age;
        String name;
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter your name : ");
        name = sc.nextLine();
        age = setAge();

        System.out.println("Name : " + name);
        System.out.println("Age : " + age);
    }

}
```

Output:



Correct Input



Incorrect Input - Exception

Conclusion :

Hence, I learnt about the ~~Exc~~ concept of exception handling. I also created, debugged and executed java programs based on exception handling.

Practical No. 12

Aim: Create, debug and run java programs based on user defined exception.

Practical No. 12

Aim: Create, debug and run java programs based on user defined Exception.

Theory:

What are user defined exceptions?

→ In Java, we can create our own exceptions that are derived classes of the Exception classes.

- Creating our own exception is known as custom exception or user defined exception.
- Basically, the custom exceptions in Java are used to customize the exception according to user need.

Why should we use custom exeptions?

→

- To catch and provide specific treatment to a subset of existing java exceptions.
-

Syntax:

```
public class className extends Exception {

    public className ( parameter ) {
        Statement(s);
    }
}
```

Conclusion:

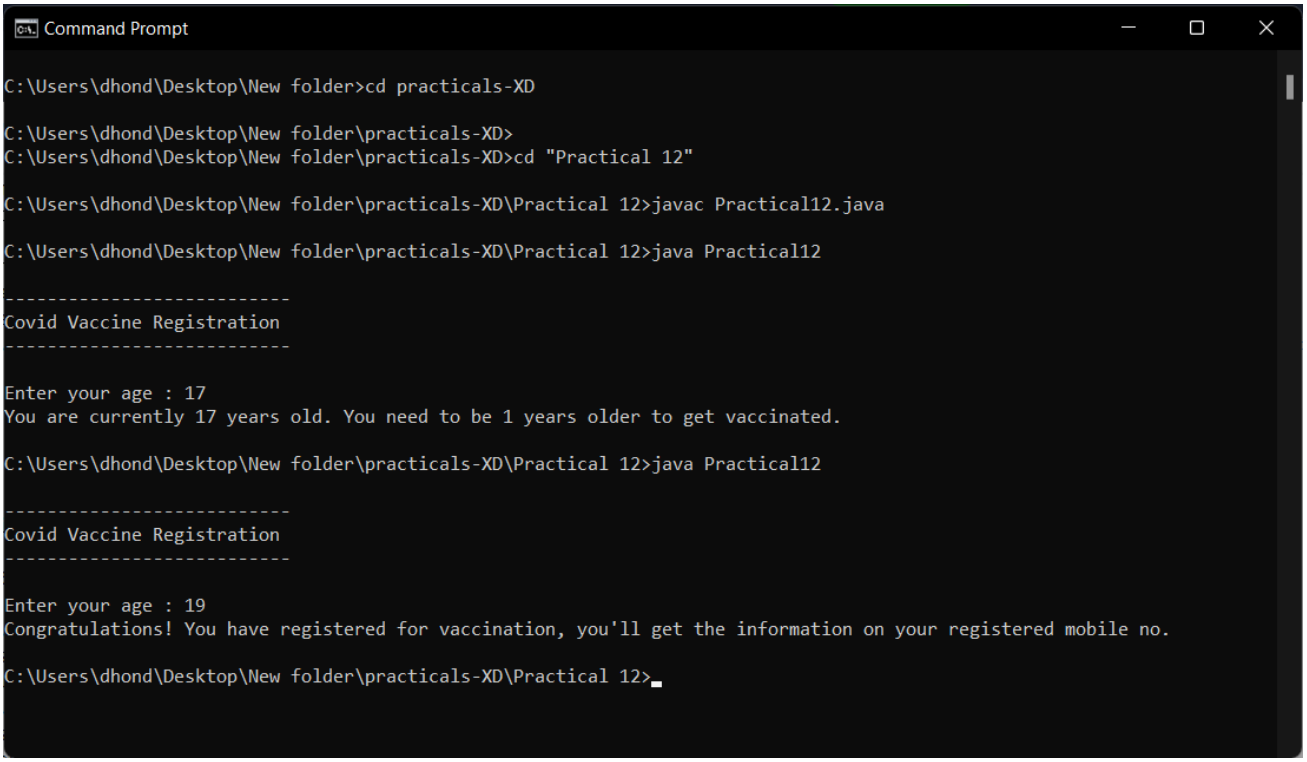
Hence, I learnt about the concept of user defined exception handling and developed, debugged and executed programs based on the concept.

Code:

```
import java.util.InputMismatchException;
import java.util.Scanner;

class InvalidAgeForVaccinationException extends Exception{
    InvalidAgeForVaccinationException(int age){
        int requiredYears = 18 - age;
        System.out.println("You are currently " + age + " years old. You need to be
" + requiredYears + " years older to get vaccinated.");
    }
}


class Practical12{

    public static int setAge(){
        Scanner sc = new Scanner(System.in);
        int age = 0;
        try{
            System.out.print("Enter your age : ");
            age = sc.nextInt();
        }catch(InputMismatchException e){
            System.out.println("Wrong input, please re-enter age again");
            age = setAge();
        }catch (Exception e){
            System.out.println("Unexpected error : ");
            System.out.println(e);
            System.out.println("Setting age to 0.");
        }finally {
            return age;
        }
    }


    public static void main(String[] args) {
        System.out.println("\n--------------------------\nCovid Vaccine
Registration\n--------------------------\n");
        Scanner sc = new Scanner(System.in);
        int age;
        try{
            age = setAge();
            if(age < 18){
                throw new InvalidAgeForVaccinationException(age);
            }
```

```
            System.out.println("Congratulations! You have registered for
vaccination, you'll get the information on your registered mobile no.");

        }catch(InvalidAgeForVaccinationException e){
//            System.out.println("Exception : " + e);
        }

    }

}
```

Output:



Conclusion :

Hence, I learnt about the concept of user defined exception handling and developed, debugged and executed programs based on the concept.

Practical No. 13

Aim : Create, debug, and run java programs based on Threads by implementing Runnable interface.

Practical No. 13

PAGE NO.
DATE

Aim: Create, debug and run java programs based on Threads by extending implementing Runnable interface.

Theory:

What is runnable interface?

- java.lang.Runnable is an interface that is to be treated by a class whose instances are intended to be executed by a thread.
- Runnable interface has only one function it in it, which has to be overidden in the class that implements Runnable interface. This function is the run public void run() function.
- Runnable interface is used in conditions where a class has to extend another class, as java doesn't support extending multiple classes, so in such cases Runnable interface is used.

What are the steps to create a Runnable class by implementing Runnable interface?

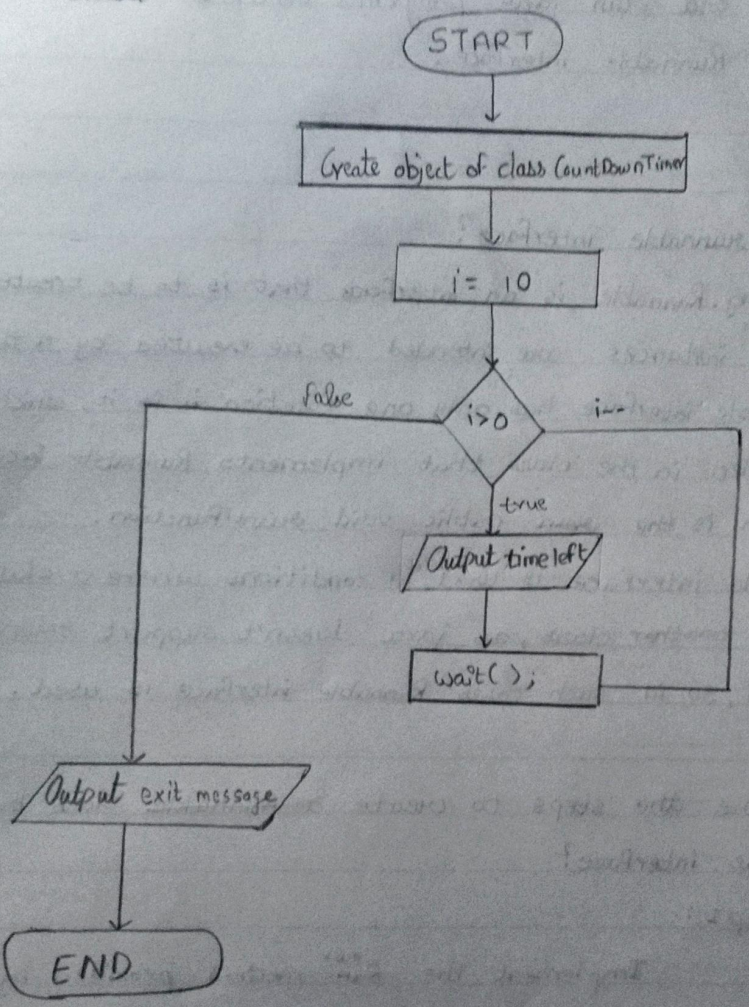
→ Step 1:

Implement the run method provided by the Runnable interface. This method provides entry point for the thread and you will the code part for the thread goes in it.

Syntax:

```
public void run(){
....
}
```

Flowchart :



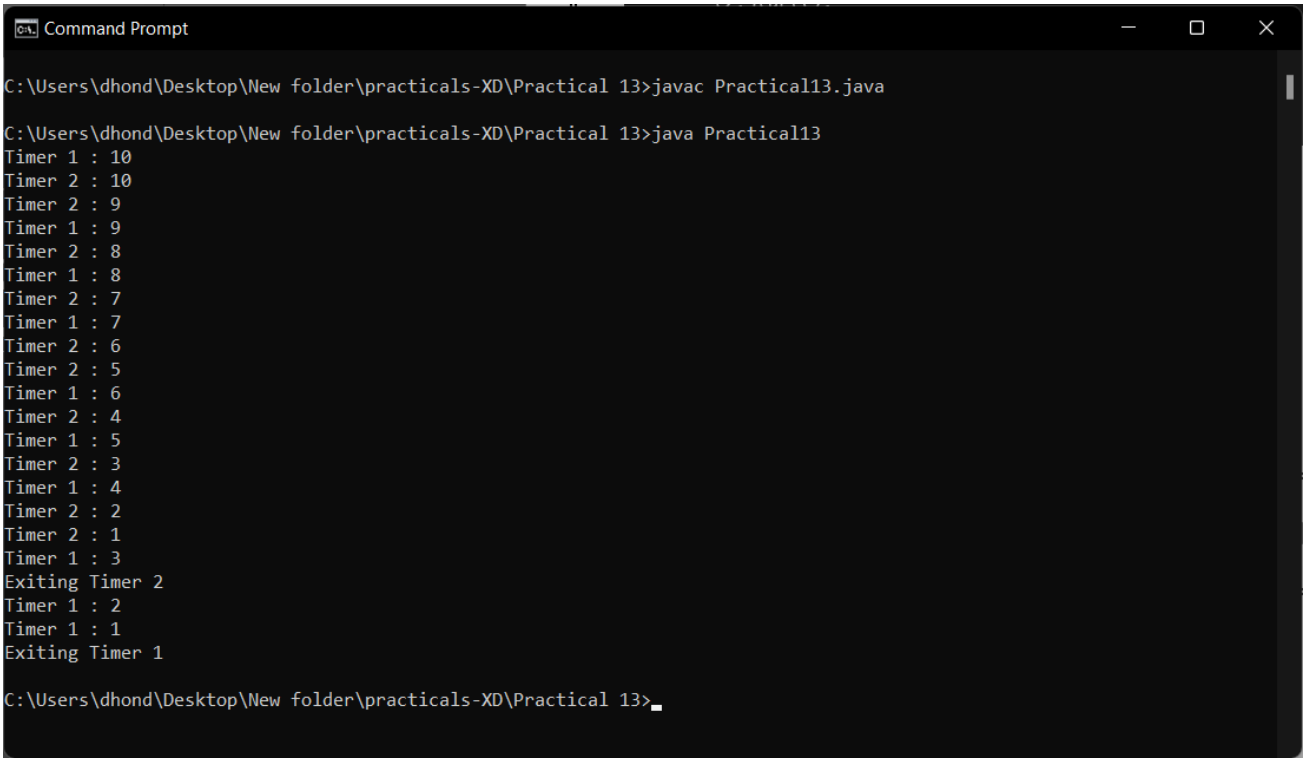
Conclusion :

Hence, by performing this practical I learnt about the concept of creating threads by using Runnable interface. I also created, developed and executed java programs based on threads by implementing Runnable interface.

Code:

```
class CountdownTimer implements Runnable{
        String name;
        int ms;

        CountdownTimer(String name, int ms){
            this.name = name;
            this.ms = ms;
        }

        synchronized public void run() {
            for(int i = 10; i > 0; i--){
                System.out.println(name + " : " + i);
                try {
                    wait(ms);
                }catch (Exception e){
                    System.out.println(e);
                }
            }
            System.out.println("Exiting " + name);
        }
    }

class Practical13{

    public static void main(String[] args) {
        // 250, 500 are milliseconds for wait
        CountdownTimer c1 = new CountdownTimer("Timer 1",1000);
        CountdownTimer c2 = new CountdownTimer("Timer 2",750);

        Thread t1 = new Thread(c1);
        Thread t2 = new Thread(c2);
        t1.start();
        t2.start();

    }

}
```

Output:


Command Prompt
C:\Users\dhond\Desktop\New folder\practicals-XD\Practical 13>javac Practical13.java
C:\Users\dhond\Desktop\New folder\practicals-XD\Practical 13>java Practical13
Timer 1 : 10
Timer 2 : 10
Timer 2 : 9
Timer 1 : 9
Timer 2 : 8
Timer 1 : 8
Timer 2 : 7
Timer 1 : 7
Timer 2 : 6
Timer 2 : 5
Timer 1 : 6
Timer 2 : 4
Timer 1 : 5
Timer 2 : 3
Timer 1 : 4
Timer 2 : 2
Timer 2 : 1
Timer 1 : 3
Exiting Timer 2
Timer 1 : 2
Timer 1 : 1
Exiting Timer 1
C:\Users\dhond\Desktop\New folder\practicals-XD\Practical 13>

PAGE NO.
DATE

Step 2:

Instantiate the object of thread using the following constructor.

Thread (Runnable threadObj);

example,

Thread t = new Thread (new ClassRunnable());

~~Class Run~~

Step 3 :

Once the thread is created, you can start it by calling the start method, which executes call to run method.

example.

t.start();

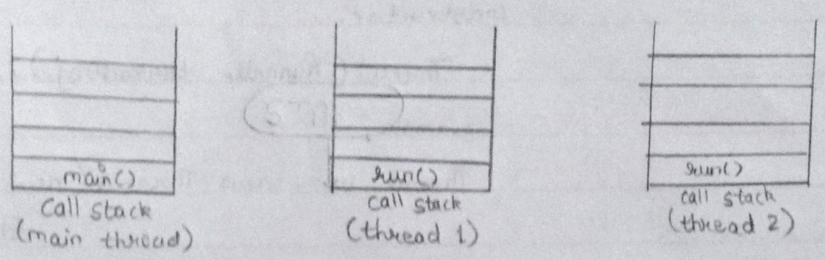
Conclusion:

Hence, by performing this practical I learnt about the concept of creating threads & by using Runnable interface. I also created, developed and executed java program based on Threads by implementing Runnable interface.

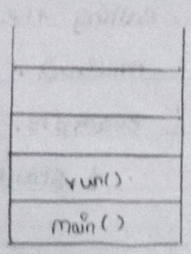
Practical No. 14

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.

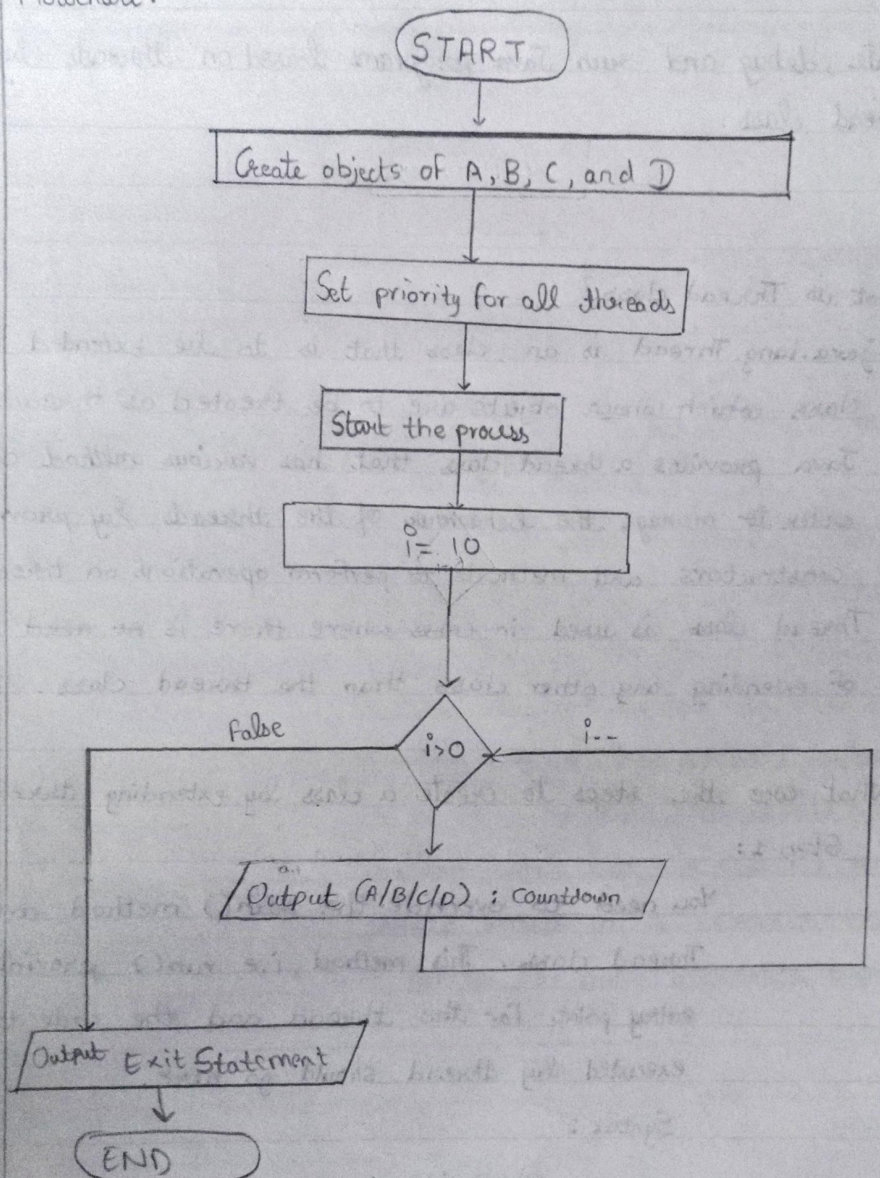
Syntax :

```
@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:

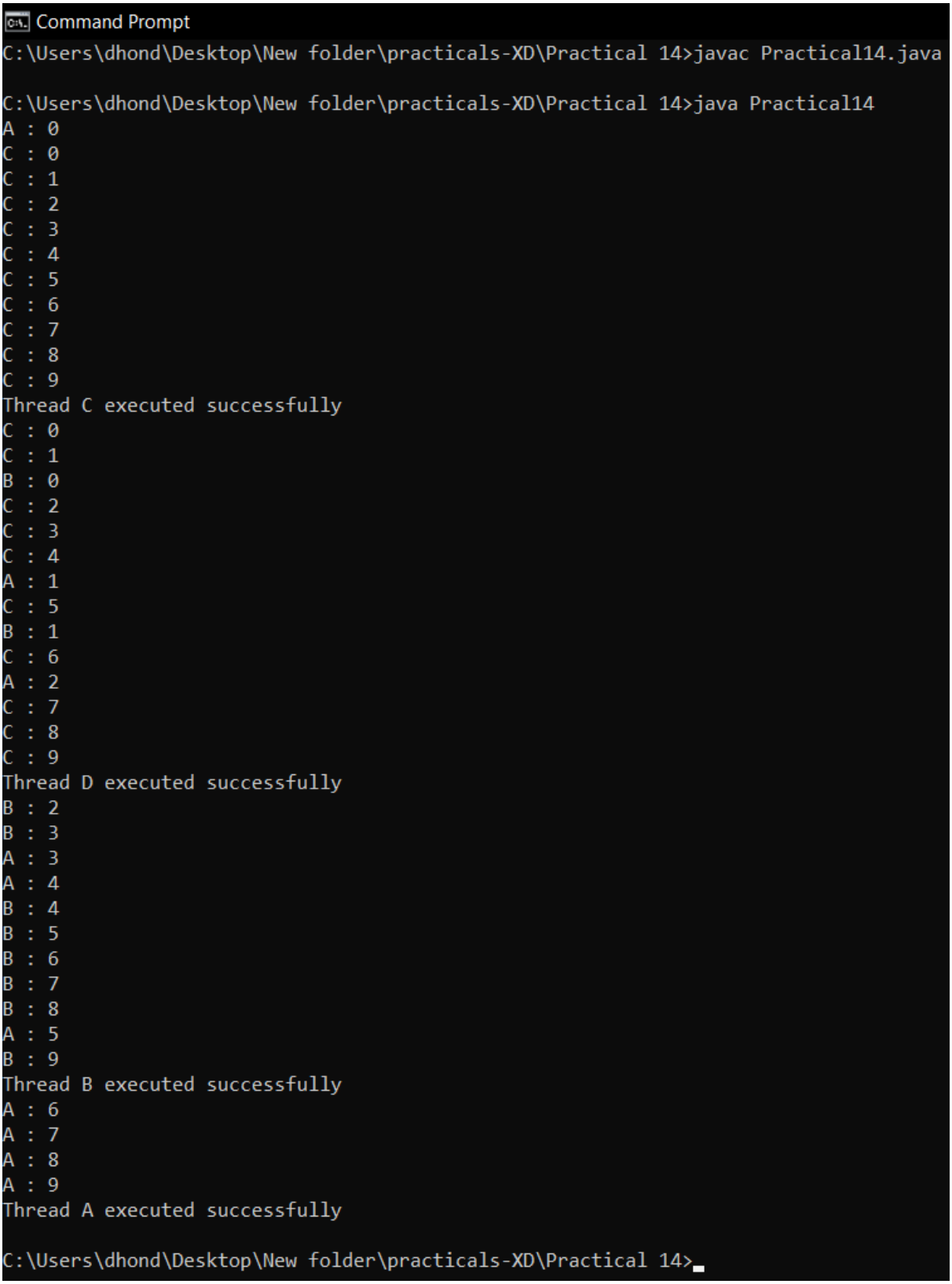
Hence, by performing this practical I got to learn about how to create java programs which can perform multithreading by extending thread classes. I also created, debugged and executed java programs based on the concept of threads by extending classes.

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.