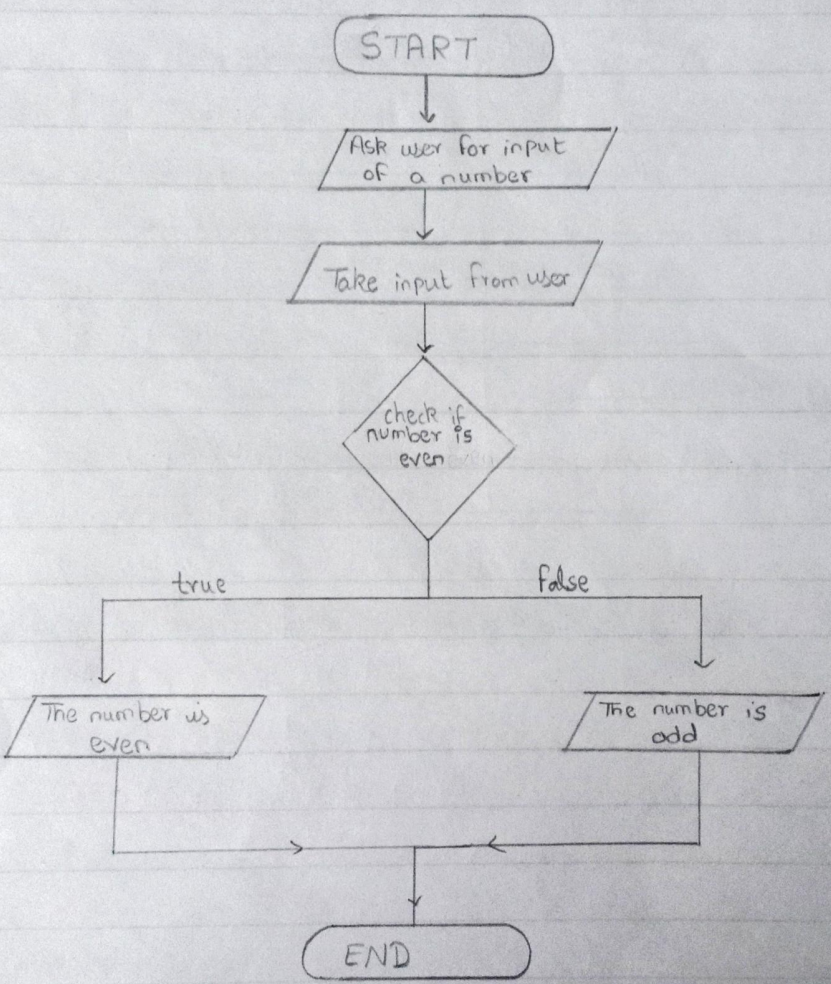
Practical No. 02

Aim: Create, debug and run java programs based on decision making and branching.

Flowchart:



Flow chart : code 1

Practical No. 02

Aim: Create, debug and run java programs based on decisions making and branching.

Theory:

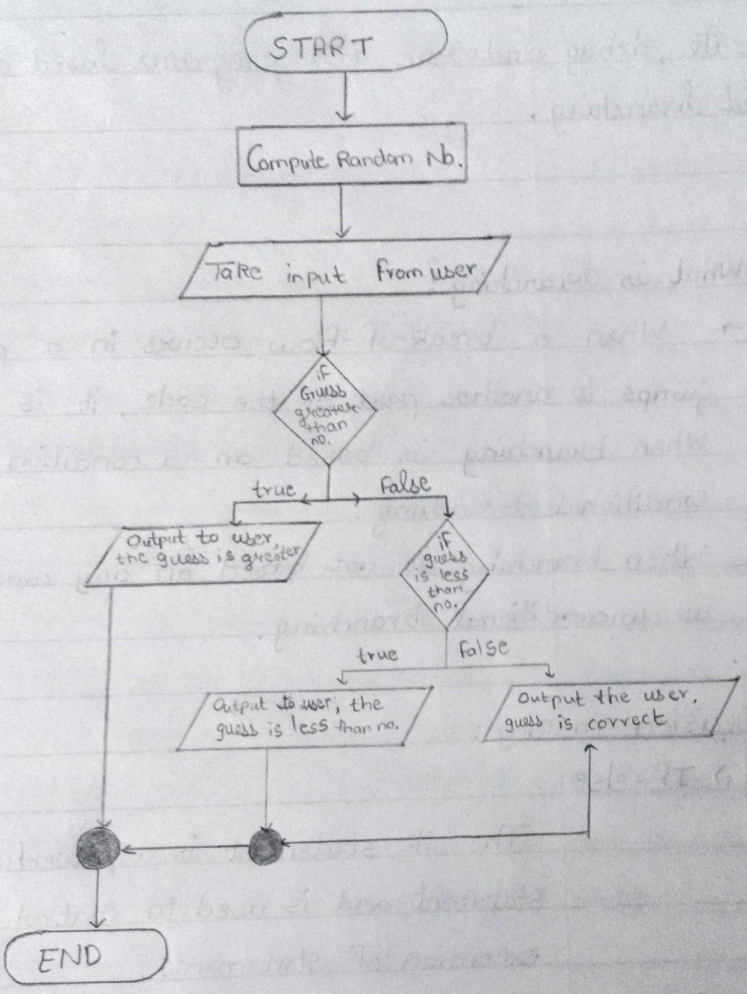
What is branching?

→ When a break-of-flow occurs in a program and it jumps to another part of the code, it is called as branching. When branching is based on a condition, it is known as conditional branching.
When branching is not based on any conditions, it is known as unconditional branching.

Decision making:

1) If-else

The if statement is a powerful decision making statement and is used to control the flow of execution of statements.
The if-else statement is the extension of the simple if statement.
If the test expression is true, then the true-block statement(s) immediately following if statement are executed.
Otherwise, the false block's statement(s) are executed.



Flow chart : code 2

Syntax:

```
if (condition){
  // code statements
}else{
  // code statements
}
```

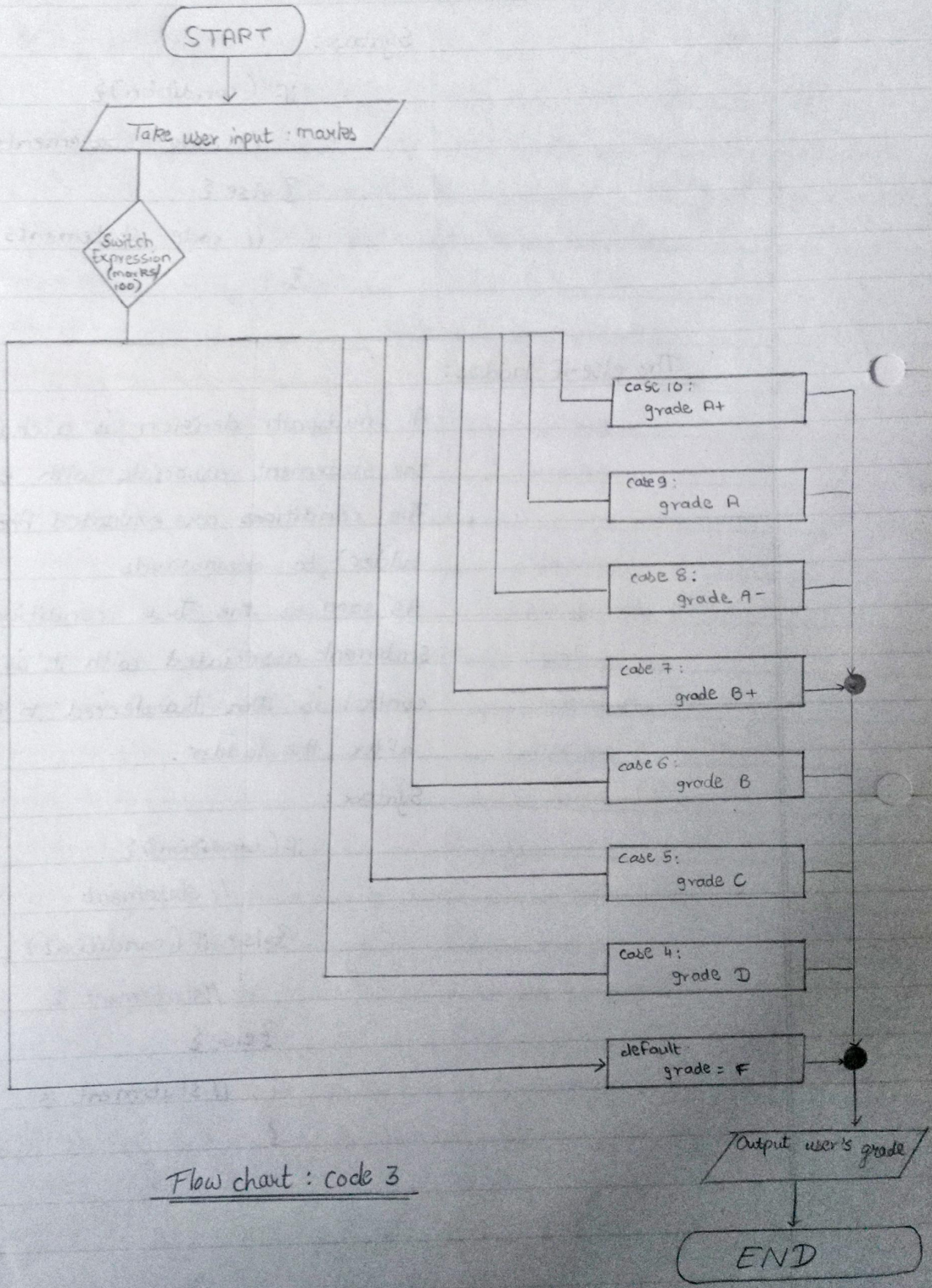
The else-if ladder:

A multipath decision is a chain of ifs in which the statement associate with each else is an if. The conditions are evaluated from the top (of the ladder), to downwards.

As soon as the true condition is found, the statement associated with it is executed and the control is ~~tra~~ transferred to the statement after the ladder.

Syntax:

```
if(condition){
  // statement
}else if(condition2){
  //statement 2
}else{
  // statement 3
}
```



Flow chart : code 3

The switch statement :

The switch statement tests the value of a given variable, against a list of case values and when a match is found, a block of statements associated with it is executed.

Syntax :

```
switch (expression) {
    case 1 :
        statement 1;
        break;
    case 2 :
        statement 2;
        break;
    default :
        statement 3;
        break;
}
```

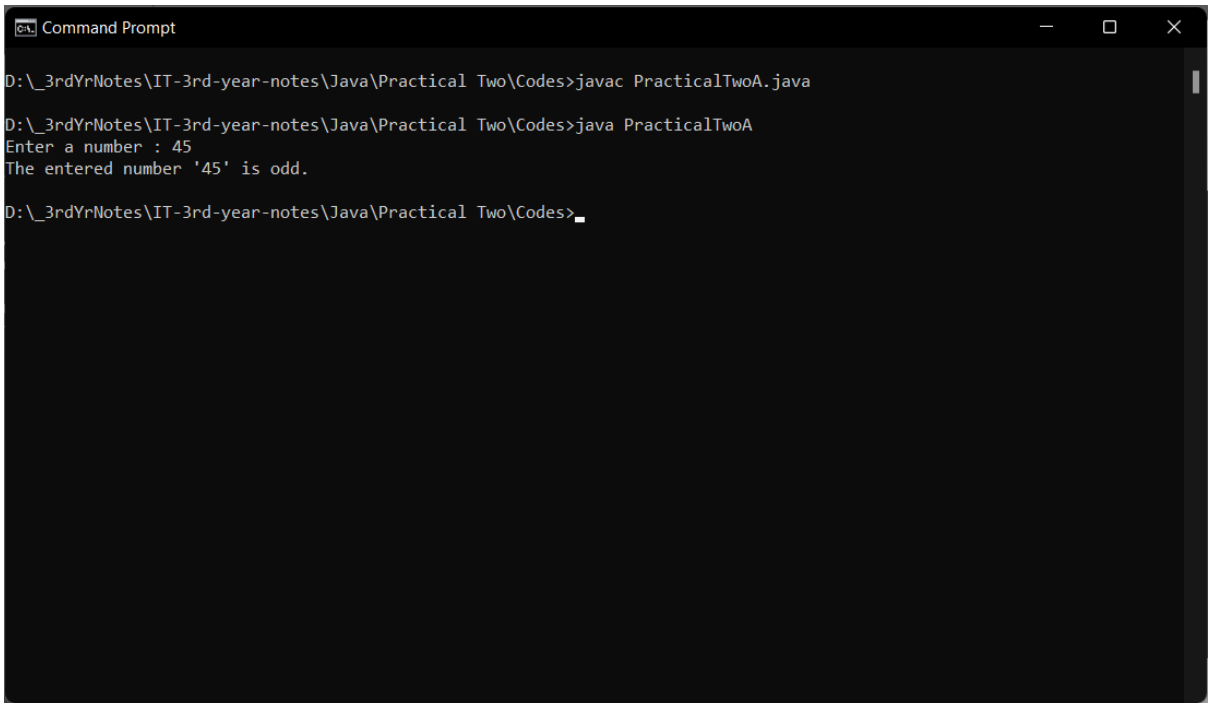
Conclusion :

Hence, by performing this practical, I learnt about decision making, and branching. I also created, debugged and executed three java programs on the concepts of if-else, if-else ladder, and switch respectively.

Code :

```java
import java.util.Scanner;

class PracticalTwoA{

  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a number : ");
    int number = sc.nextInt();
    if(number%2 == 0){
      System.out.println("The entered number '" + number + "' is even.");
    }else{
      System.out.println("The entered number '" + number + "' is odd.");
    }
  }

}
```

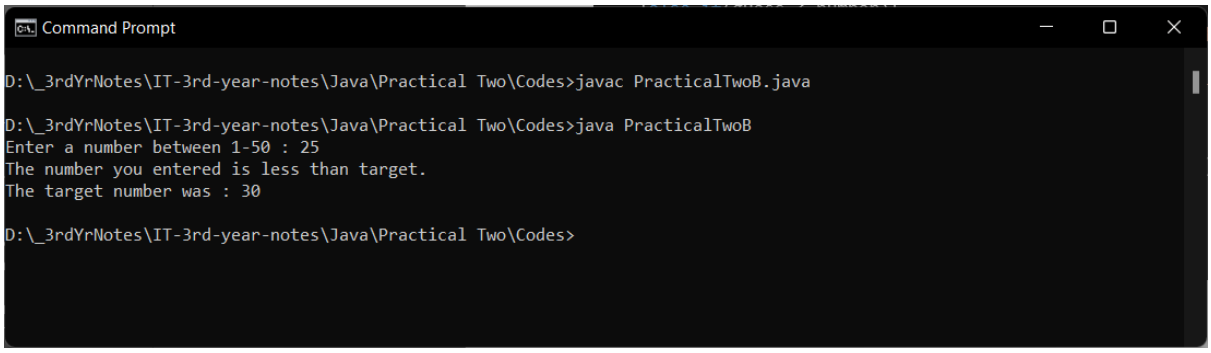
Output :



Code :

```java
import java.util.Scanner;
import java.util.Random;

class PracticalTwoB{

  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    Random rand = new Random();
    int number = rand.nextInt(50) + 1;
    int guess;
    if(args.length == 0){
    System.out.print("Enter a number between 1-50 : ");
    guess = sc.nextInt();
    }else{
      guess = Integer.parseInt(args[0]);
    }

    if(guess > number){
      System.out.println("The number you entered is greater than target.");
    }else if(guess < number){
      System.out.println("The number you entered is less than target.");
    }else{
      System.out.println("The number you entered is equal to Target!");
    }

    System.out.println("The target number was : " + number);

  }

}
```

Output:

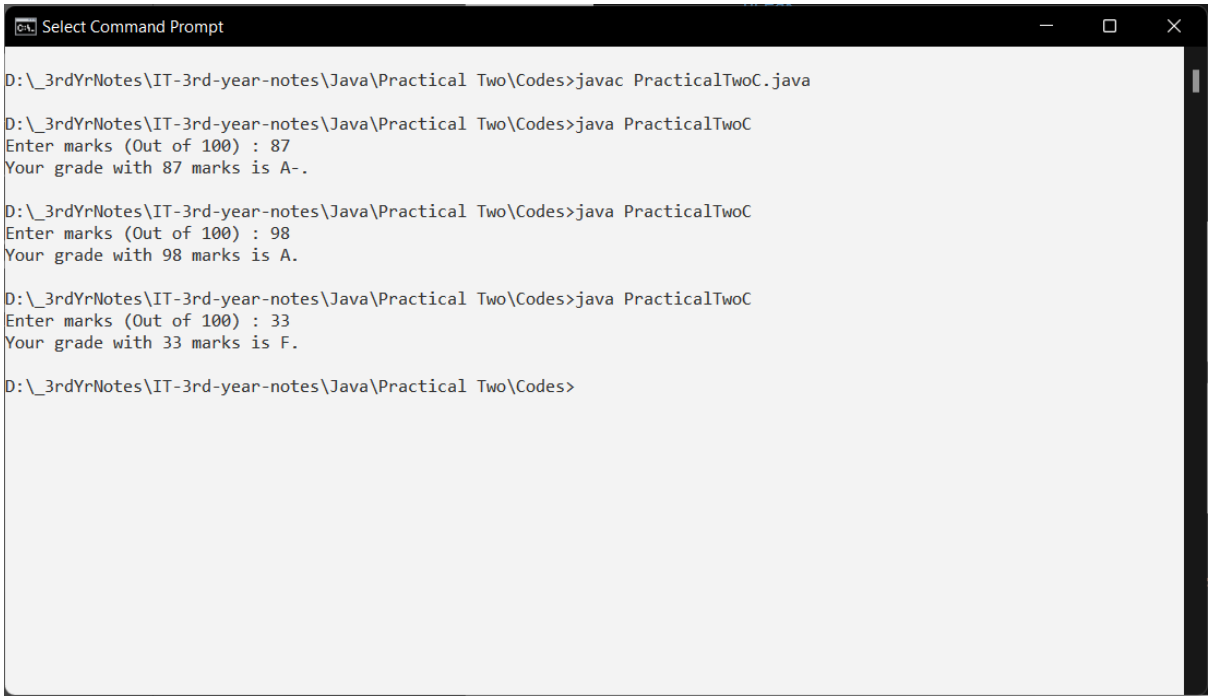


Code :

```
import java.util.Scanner;

class PracticalTwoC{

  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    int marks = 0;
    if(args.length == 0 ){
      System.out.print("Enter marks (Out of 100) : ");
      marks = sc.nextInt();
    }else{
      try{
        marks = Integer.parseInt(args[0]);
      }catch(Exception e){
        marks = 0;
        System.out.println(e.toString());
      }
    }
    String grade;

    switch(marks/10){
      case 10:
              grade = "A+";
              break;
      case 9:
              grade = "A";
              break;
      case 8:
              grade = "A-";
              break;
      case 7:
              grade = "B+";
              break;
      case 6:
              grade = "B";
              break;
      case 5:
              grade = "C";
              break;
      case 4:
              grade = "D";
              break;
      default:
              grade = "F";
              break;
    }
```

```
 System.out.println("Your grade with " + marks + " marks is " + grade + ".");

  }

}
```

Output:



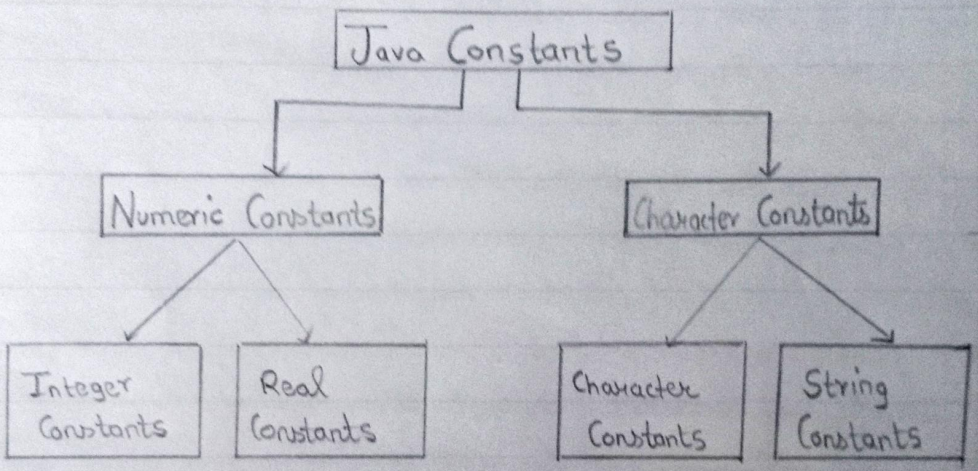
Conclusion:

Hence, by performing this practical, I learnt about decision making and branching. I also created, debugged and executed three java programs on the concepts of if-else, if-else ladder, and switch ~~op.~~ respectively.

Practical No. 01

Aim: Create, debug and run java programs based on constants, variables and operators.

Diagram:



Java Constants

Practical No. 01

Aim : Create, debug and run java programs based on constants, variables and operators.

Theory :

1) Constants :

- Constants in Java refer to fixed values that do not change during execution of programme.
- Constants in Java are divided into two sub-types, numeric constants and character constants.
- The numeric constants are furthur divided into integer constants and real constants.
- The character constants are furthur divided into character constants and string constants.
- Examples,

numeric constants :

i) Integer constants - 123, -321

ii) Real constants - 0.083, -0.15, 0.65E4

character constants :

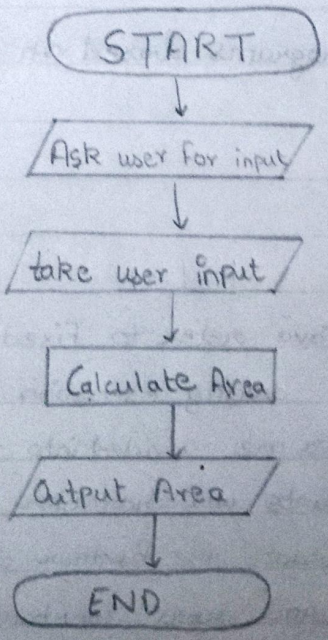
i) character constants - 'x', '5', 'A'

ii) String constants - "Hello", "1997", "?.$.."

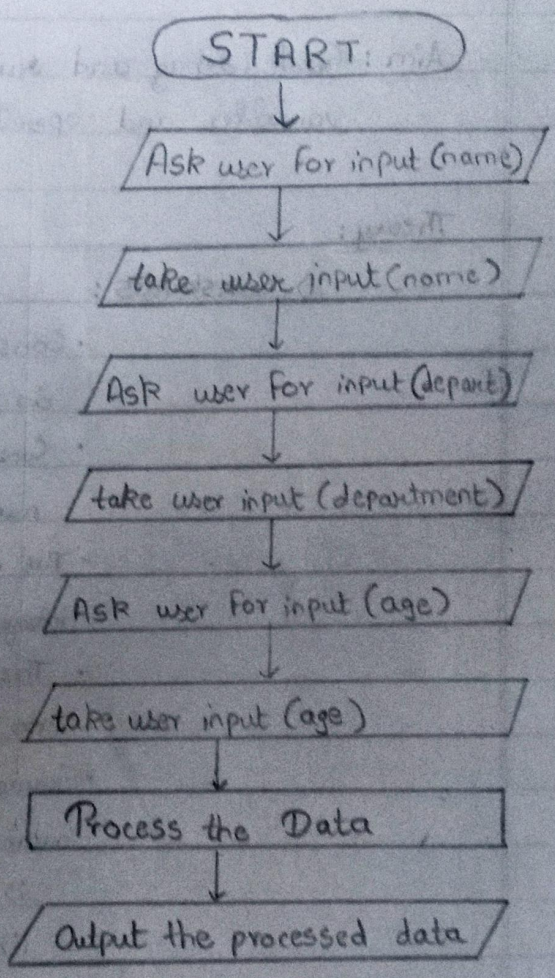
2) Variables :

- A variable is an identifier that denotes a storage location used to store a data value.
- Unlike constants that remain unchanged during the execution of a program, a variable may take different values at different times during the

Flowchart:



Code 1 Flow chart:
Constants



Code 2 Flow chart: Operator

execution of the program.

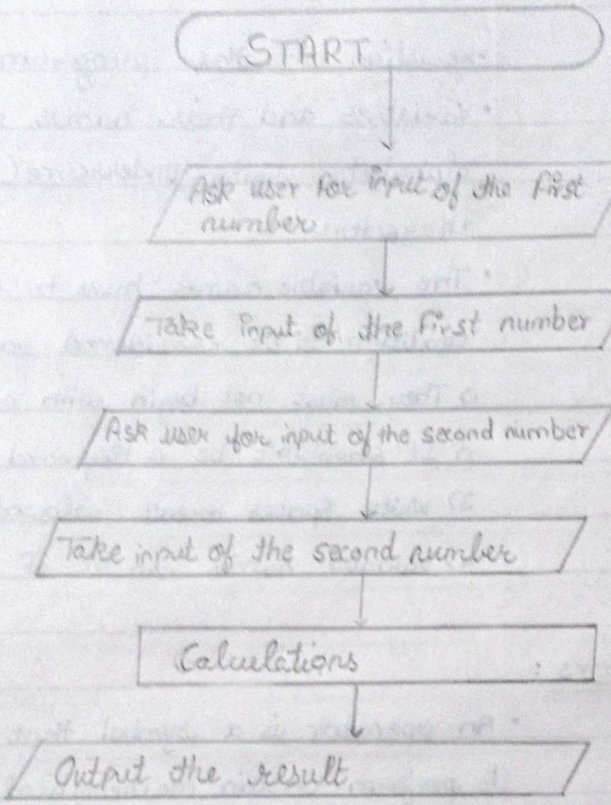
- Variables and their names may consist of alphabets, digits, underscore (_), and dollar ($) characters.
- The variable names have to meet the below condition to be considered valid.

1) They must not begin with a digit.
2) It shouldn't be a Keyword.
3) White spaces aren't allowed.
4) Variable names can be of any length.

3) Operators :

- An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.
- The java operators can be specified into a number of related categories as below.

1) Arithmetic operators, e.g. +, -, /, *, %.
2) Relational operators, e.g. <, >, ==, !=
3) Logical operators, e.g. &&, ||, !
4) Assignment operators, e.g. =
5) Increment / Decrement operators. ++, --
6) Conditional operators, ?:
7) Bitwise operators, &, |, ^, ~, <<, >>>, >>
8) Special operators.



Code & Flow chart : Operators

Conclusion :

Hence, by performing this practical, I learnt about the concepts of constants, variables and operators in Java. I also created, debugged and executed three Java programs on the concepts of constants, variables and operators respectively.

Code 1 : Constants

```java
import java.util.Scanner;

class PracticalOneA {
  public static void main(String[] args) {
    final double PI = 3.14;
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter Radius of Circle : ");
    float radius = sc.nextFloat();
    System.out.println("Area of circle : " + (PI * radius * radius) );

    System.out.print("Enter length and breadth of Rectangle : ");
    float length = sc.nextFloat();
    float breadth = sc.nextFloat();
    System.out.println("Area of Rectangle : " + (length * breadth) );

    System.out.print("Enter Side of Square : ");
    float side = sc.nextFloat();
    System.out.println("Area of Square : " + (side * side) );

  }
}
```

Output :

```
D:\_3rdYrNotes\IT-3rd-year-notes\Java\Practical One\codes>javac PracticalOneA.java

D:\_3rdYrNotes\IT-3rd-year-notes\Java\Practical One\codes>java PracticalOneA
Enter Radius of Circle : 10
Area of circle : 314.0
Enter length and breadth of Rectangle : 10 20
Area of Rectangle : 200.0
Enter Side of Square : 15
Area of Square : 225.0
```

Code 2 : Variables

```java
import java.util.Scanner;

class PracticalOneB {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    var name = " ";
    var age = 0;
    var department = " ";

    System.out.print("Enter your full name  : ");
    name = sc.nextLine();
    System.out.print("Enter your Department : ");
    department = sc.nextLine();
    System.out.print("Enter your age        : ");
    age = sc.nextInt();

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


  }
}
```

Output :

```
D:\_3rdYrNotes\IT-3rd-year-notes\Java\Practical One\codes>java PracticalOneB
Enter your full name  : Pratyay Prasad Dhond
Enter your Department : Information Technology
Enter your age        : 18
Name       : Pratyay Prasad Dhond
Department : Information Technology
Age        : 18
```

Code 3 : Operators

```java
import java.util.Scanner;

class PracticalOneC {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int a;
    int b;

    System.out.print("Enter First Number : ");
    a = sc.nextInt();
    System.out.print("Enter Second Number : ");
    b = sc.nextInt();

    System.out.println(a + " + " + b + " = " + (a + b ));

    System.out.println(a + " - " + b + " = " + (a - b ));

    System.out.println(a + " * " + b + " = " + (a * b ));

    System.out.println(a + " / " + b + " = " + (a / b ));

    System.out.println(a + " % " + b + " = " + (a % b ));


  }
}
```

Output :

```
D:\_3rdYrNotes\IT-3rd-year-notes\Java\Practical One\codes>java PracticalOneC
Enter First Number : 18
Enter Second Number : 5
18 + 5 = 23
18 - 5 = 13
18 * 5 = 90
18 / 5 = 3
18 % 5 = 3
```

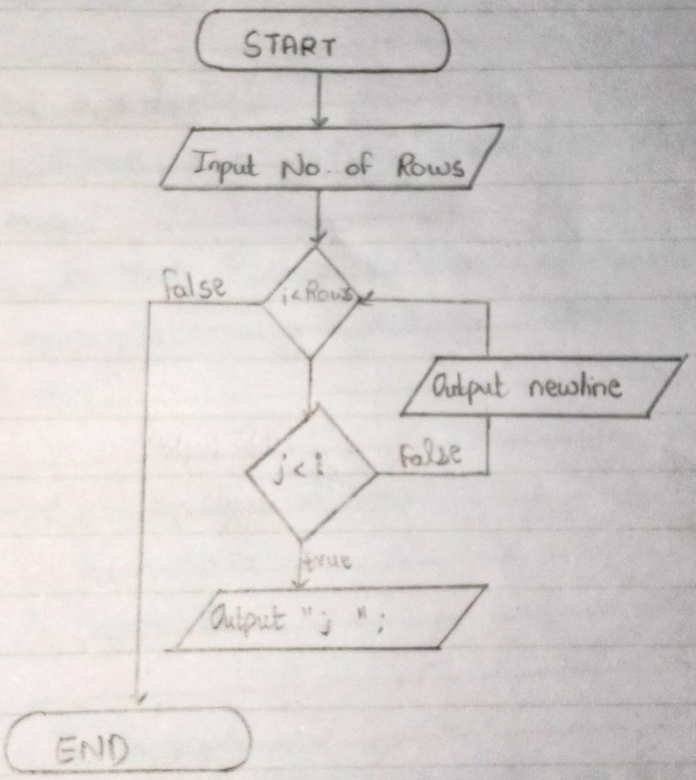
Conclusion:

Hence, by performing this practical, I learnt about the concepts of constants, variables and operators in Java. I also created, debugged and executed three Java programs on the concepts of constants, variables and operators respectively.

Practical No. 03

Aim: Create, debug, and run java programs based on decision making and looping.

Flow chart:



Code 1

Practical No. 03

Aim: Create, debug, and run java programs based on decision making and looping.

Theory:

What is looping?

→ A loop is a way of representing lines of code more than once.

The block of code contained within the loop will be executed again and again until the condition required by the loop is met.

What are the types of loops?

→ i) Indeterminate
ii) Determinate

i) Indeterminate:

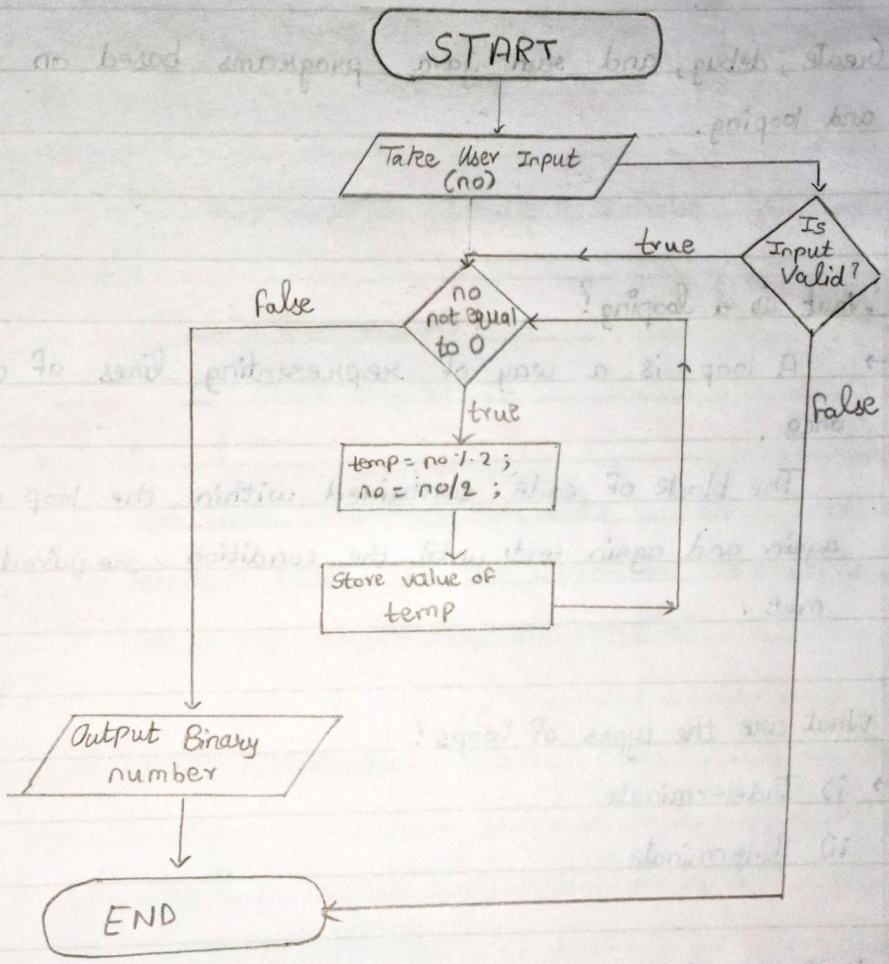
An indeterminate loop does not know how many times it will run.

For example, while, do...while.

ii) Determinate:

A determinate loop knows exactly how many times it will loop.

For example, for loop.


START
Take User Input (no)
Is Input Valid?
true
false
false
no not equal to 0
true
temp = no % 2 ;
no = no/2 ;
Store value of temp
Output Binary number
END

Code 2

The While and do-while statements:

The while statement continually executes a block of statements while a particular condition is true. Its syntax can be expressed as:

```
while (expression) {
    statement(s)
}
```

The while statement evaluates expression, which must return a boolean value. If the expression evaluates to ~~tre~~ true, the while statement execute the statement(s).

Do-while statement:

```
do {
    statement(s)
} while(expression)
```

The difference between do-while and while is that do-while evaluates its expressions at the bottom of the loop instead of the top.

The for statement :

The for statement provides a compact way to iterate over a range of values. Programmers often refer to it as the "for loop" because of the way in which it repeatedly loops until a particular condition is satisfied.

```
for (initialization ; termination ; increment) {
        statement(s);
}
```

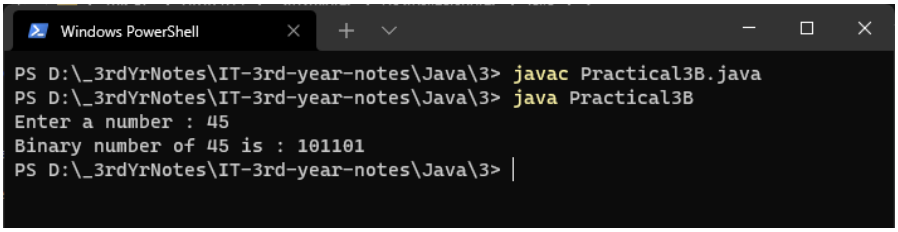
- The initialization expression initializes the loop ; its executed once as it begins.
- When the termination expression evaluates to false, the loop terminates.
- The increment expression is invoked after each iteration through the loop ; it is perfectly acceptable for this expression to increment or decrement a value.

Conclusion : Hence, by performing this practical I learnt the concept of decision making and looping and I also created, debugged and ran java programs based on decision making and looping

Code:

```java
// package com.practicals;

import java.util.Vector;
import java.util.Scanner;

public class Practical3B {

    public static void main(String[] args) {
        Vector<Integer> binary = new Vector<Integer>();
        Scanner sc = new Scanner(System.in);
        int no;

        System.out.print("Enter a number : ");
        try {
            no = sc.nextInt();
        }catch (Exception e){
            System.out.println("Invalid Input");
            System.out.println("Exitting Program...");
            return;
        }

        int num = no;
        while(no != 0){
            int temp = no%2;
            no = no/2;
            binary.add(temp);
        }

        System.out.print("Binary number of " + num + " is : ");
        for(int i = binary.size()-1;i>=0;i--){
            System.out.print(binary.elementAt(i));
        }

    }
}
```

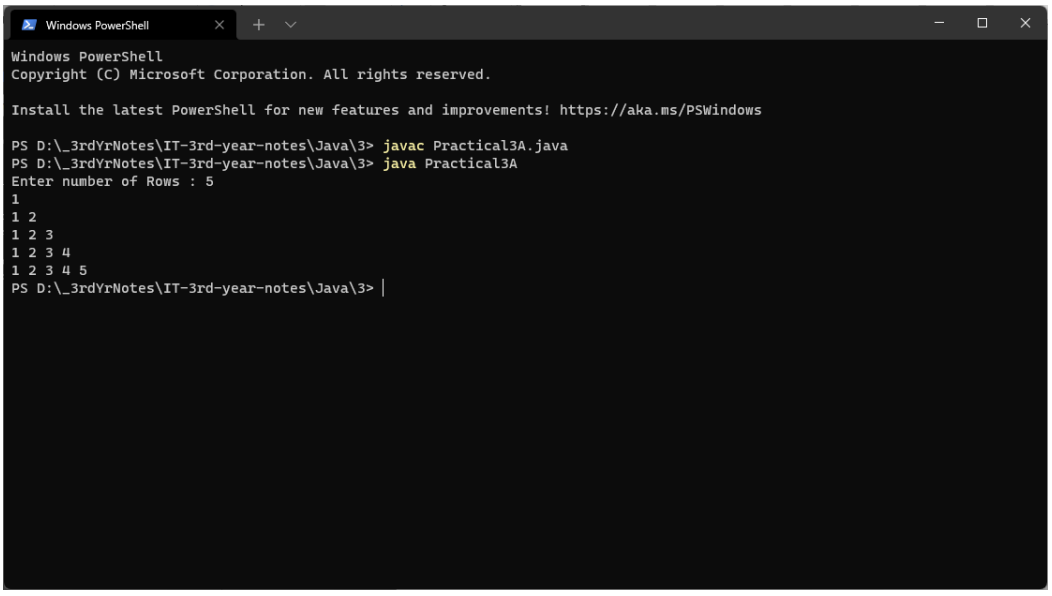
Output:



Code:

```java
import java.util.Scanner;


class Practical3A{

  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    int no;
    if(args.length == 0){
      System.out.print("Enter number of Rows : ");
      no = sc.nextInt();
    }else{
      try{
      no =  Integer.parseInt(args[0]);
      }catch(Exception e){
        no = args[0].length();
      }
    }

    for(int i = 1; i<=no ; i++){
      for(int j = 1; j<=i; j++){
        System.out.print(j + " ");
      }
      System.out.println();
    }
  }

}
```

Output:



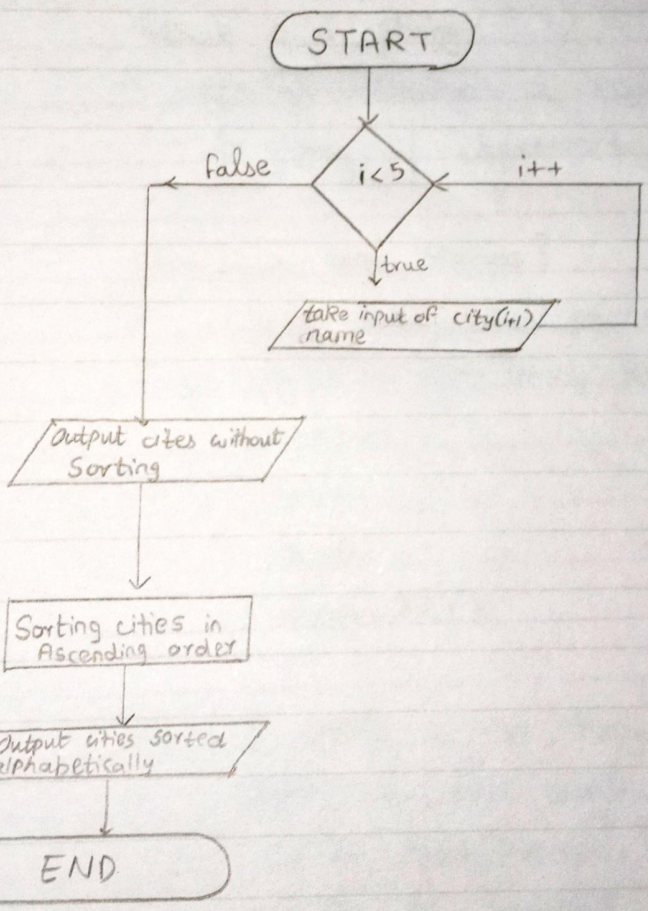
Conclusion:

Hence, by performing this practical I learnt the concept of decision making and looping, and created, debugged and ran java programs based on decision making and looping.

Practical No. 04

Aim: Create, debug and run programs based on String and StringBuffer.

Flow chart:



Code 1

Practical No. 04

Aim: Create, debug and run programs based on String and StringBuffer.

Theory:

What is a String?

→ • Strings represent a sequence of characters.
• An array of characters works same as Java string.

Why use Java Strings?

→ Java String class provides a lot of methods to perform operations on ~~str~~ string such as: compare().

- compare()
- concat()
- length()
- compareTo()
- equals()
- split()
- replace()
- substring()

How to create a new string object?

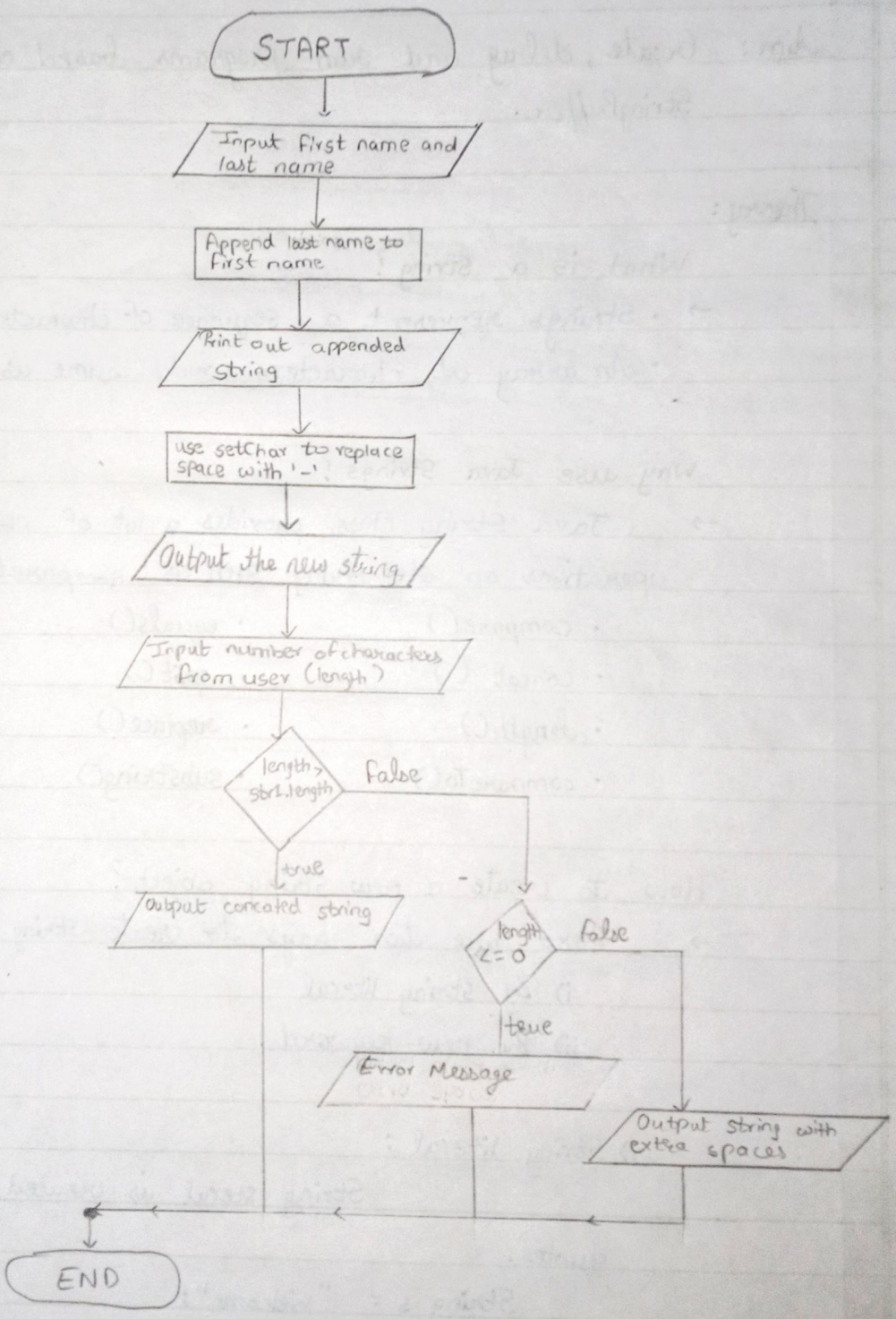
→ There are two ways to create String object:

i) By string literal
ii) By new keyword.

i) String literal:

String literal is created by using double quotes.

```
String s = "Welcome";
String str = "Pratyay";
```


START
Input First name and last name
Append last name to first name
Print out appended string
use setChar to replace space with '-'
Output the new string
Input number of characters from user (length)
length > str1.length
false
true
output concated string
length <= 0
false
true
Error Message
Output string with extra spaces
END

Code 2

ii) By new keyword :

```
String s = new String("Pratyay");
```

In this case, the Java Virtual Machine will create a new ~~strin~~ string object in ~~norma~~ ~~no~~ normal heap memory, and the literal "~~Welcome~~" "Pratyay" will be placed in the string constant pool. The variable s will refer to the object in a heap.

StringBuffer class :

- StringBuffer is a peer class of String.
- While String creates of fixed length, StringBuffer creates String of ~~flexing~~ flexible length that can be modified in terms of both length and content.

Constructor	Description
StringBuffer()	It creates an Empty String buffer with initial capacity of 16
StringBuffer(String)	It creates a String buffer with the specified string.
StringBuffer(int)	It creates an empty String buffer with the specified capacity as length

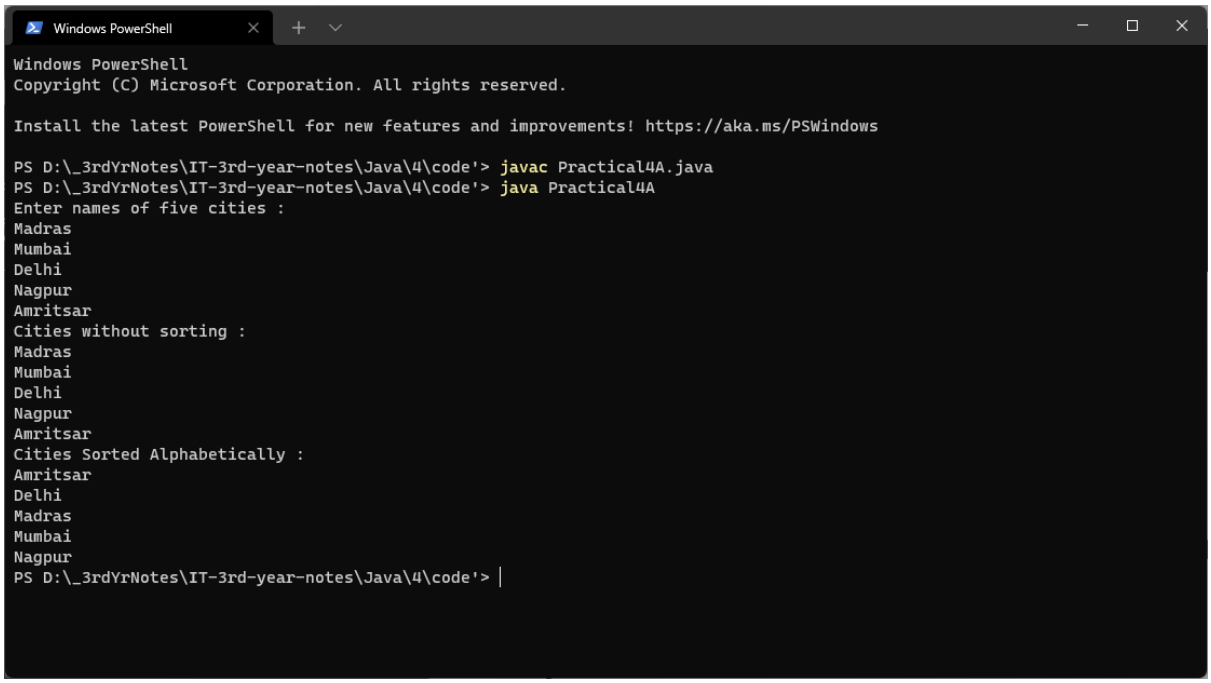
Conclusion:

Hence, I created, debugged and executed programs based on String and StringBuffer. I also learnt about the concepts of Strings and StringBuffer.

Code:

```
// String

import java.util.Scanner;

class Practical4A {

    public static void main(String[] args){
        String city[] = new String[5];
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter names of five cities : ");
        for(int i = 0;i<5;i++){
            String temp = sc.nextLine();
            city[i] = temp;
        }

        System.out.println("Cities without sorting : ");
        for(String a : city){
            System.out.println(a);
        }

        for(int i = 0; i< city.length;i++){
            for(int j = i+1; j< city.length; j++){
                if(city[i].compareToIgnoreCase(city[j]) > 0){
                    String temp = city[i];
                    city[i] = city[j];
                    city[j] = temp;
                }
            }
        }

        System.out.println("Cities Sorted Alphabetically : ");
        for(String a : city){
            System.out.println(a);
        }

    }

}
```

Output:

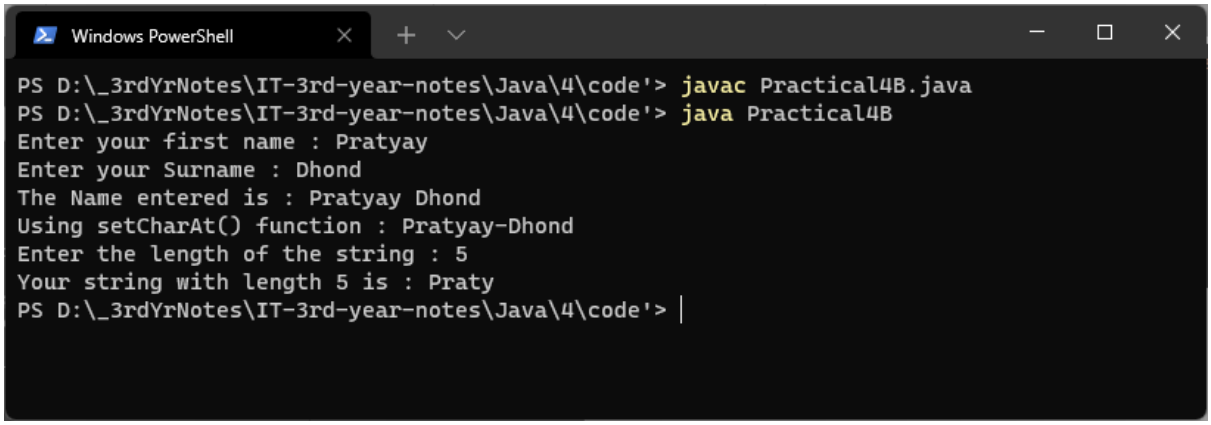

Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\4\code'> javac Practical4A.java
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\4\code'> java Practical4A
Enter names of five cities :
Madras
Mumbai
Delhi
Nagpur
Amritsar
Cities without sorting :
Madras
Mumbai
Delhi
Nagpur
Amritsar
Cities Sorted Alphabetically :
Amritsar
Delhi
Madras
Mumbai
Nagpur
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\4\code'>

Code:

```
//String Buffer

import java.util.Scanner;

class Practical4B{

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

        System.out.print("Enter your first name : ");
        //  StringBuffer str1 = new StringBuffer("James");
        temp = sc.next();
        StringBuffer str1 = new StringBuffer(temp);
        length = str1.length();
        System.out.print("Enter your Surname : ");
        //  StringBuffer str2 = new StringBuffer("Gosling");
        temp = sc.next();
        StringBuffer str2 = new StringBuffer(temp);

        // Append function and insert function of String buffer class
        System.out.println("The Name entered is : " +
str1.append(str2).insert(length,' '));

        //setCharAt() function of String buffer class
        str1.setCharAt(length,'-');
        System.out.println("Using setCharAt() function : " + str1);


        System.out.print("Enter the length of the string : ");
        length = sc.nextInt();

        if(length > str1.length()){
            System.out.println("The string has less characters than the
inputted length");
//            System.out.println("The extra characters will be replaced with
'0's. ");
            str1.setLength(length);
            System.out.println("Your String with length " + length + " is : "
+ str1);
        }else if(length <= 0){
            System.out.println("Length of string cannot be zero or less than
zero.");
        }else{
            str1.setLength(length);
```

```
            System.out.println("Your string with length " + length + " is : "
+ str1);
        }


    }

}
```

Output:



Important method of StringBuffer class :

- setCharAt(n, 'x')
- append(s2)
- insert(n, s2)
- setLength(n)

How to create a StringBuffer object ?

```
StringBuffer sb = new StringBuffer();
```

Name of class — Object name — Calling default constructor

Initializing StringBuffer object :

```
StringBuffer str = new StringBuffer("Lightning");
```

here, the object str will be initialized to "Lightning".

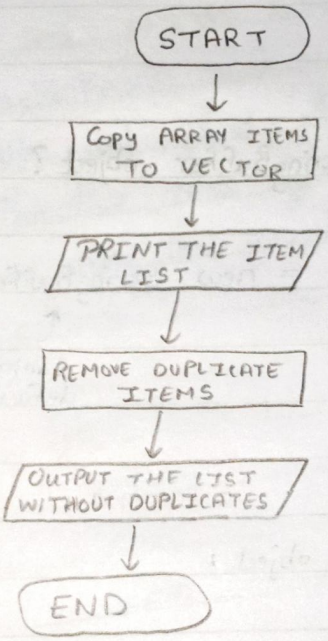
Conclusion :

Hence, I created, debugged and executed programs based on String and StringBuffer. I also learnt about the concepts of Strings and String Buffer.

Practical No. 05

Aim : Create, debug and run java programs based on Wrapper class and Vectors

Flow chart :



code 1 : VECTOR

Practical No. 05

Aim: Create, debug and ~~rerun~~ java programs based on Wrapper class and vectors.

Theory:

What are ~~wrapper~~ classes?

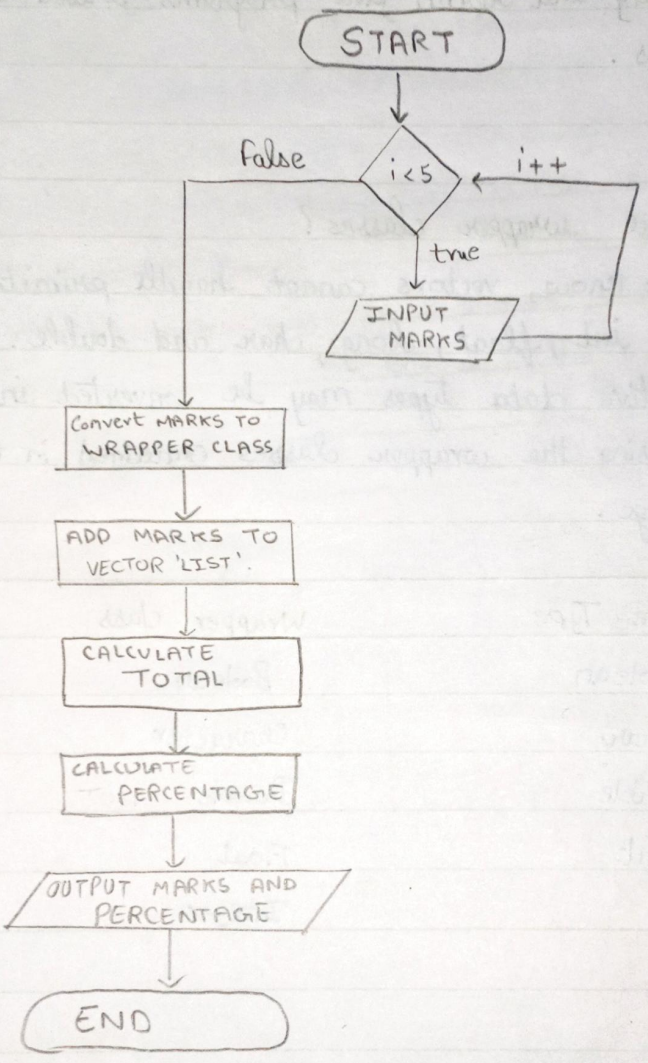
→ • As we know, vectors cannot handle primitive data types like int, float, long, char and double.
• Primitive data types may be converted into object types by using the wrapper classes contained in the java.lang package.

Simple Type	Wrapper class
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer

Autoboxing:

• The automatic conversion of primitive data type into its corresponding wrapper class is known as autoboxing.
• Since Java 5, we do not need to use the ~~valueOf()~~ valueOf() method.
• example.

```
int a = 25;
Integer b = ~~new Integer(a);~~ Integer.valueOf(a);
```



Code 2: Wrapper Class

Unboxing:

- Automatic conversion of wrapper class type into its corresponding primitive type is known as unboxing.
- This process is reverse of autoboxing.
- Example:

```
Integer a = new Integer(35);
int b = a; // now, since Java 5, compiler will write
           //a.intValue(); internally.
```

Vectors in Java:

- The Vector class in Java implements a growable array of objects.
- To use Vector class, we need to import 'java.util.Vector'.
- Vector implements a dynamic array, i.e. it can grow and shrink size according to the need.

Declaration:

```
Vector<e> v = new Vector<e>();
ex. Vector<String> v = new Vector<String>();
```

Vector methods:

```
v.addElement(item);
v.elementAt(10);
v.size();
v.removeElement(item);
v.removeElementAt(index);
```

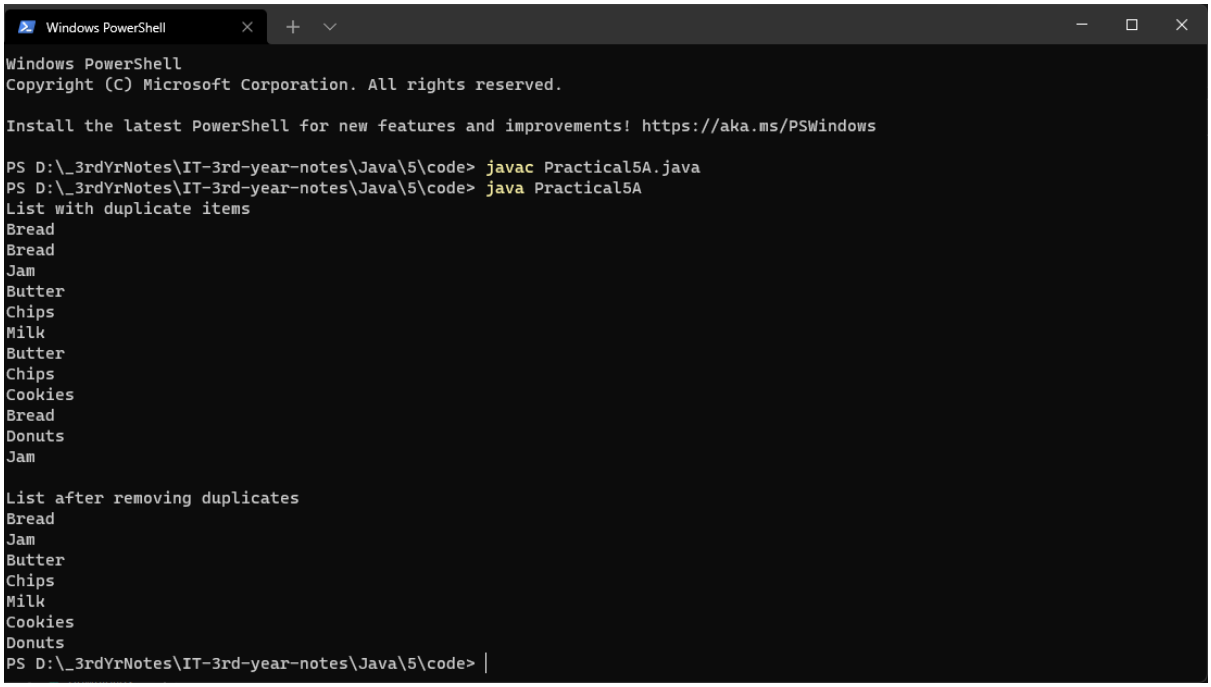
Conclusion :

Hence, by performing this practical I learnt about the concepts of Wrapper classes and Vector. I also coded, debugged and executed java programs based on the concepts of Wrapper classes and Vectors.

Code:

```
import java.util.Vector;

class Practical5A{

    public static void main(String[] args){

        Vector<String> list = new Vector<String>();
//        String list[] = new String[10];
        String items[] =
{"Bread","Bread","Jam","Butter","Chips","Milk","Butter","Chips","Cookies","Bre
ad","Donuts","Jam"};

        //copying items into the vector 'list'
        for(String a : items){
            list.add(a);
        }

        System.out.println("List with duplicate items");
        for(String a : list){
            System.out.println(a);
        }
        System.out.println();

        for(int i = 0;i<list.size()-1;i++){
            for(int j = i+1;j<list.size();){
                if(list.elementAt(i) == list.elementAt(j)){
                    list.remove(j);
                }else{
                    j++;
                }
            }
        }

        System.out.println("List after removing duplicates");
        for(String a : list){
            System.out.println(a);
        }



    }

}
```

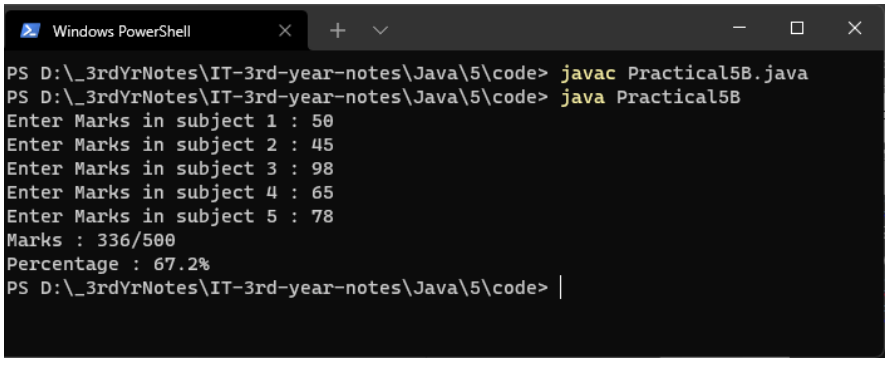
Output:


Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\5\code> javac Practical5A.java
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\5\code> java Practical5A
List with duplicate items
Bread
Bread
Jam
Butter
Chips
Milk
Butter
Chips
Cookies
Bread
Donuts
Jam
List after removing duplicates
Bread
Jam
Butter
Chips
Milk
Cookies
Donuts
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\5\code>

Code:

```java
import java.util.Scanner;
import java.util.Vector;

//@SuppressWarnings("unchecked")
class Practical5B{

    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        Vector<Integer> list = new Vector<Integer>();
        int marks;
        for(int i = 0;i < 5 ; i++){
            System.out.print("Enter Marks in subject " + (i+1) + " : ");
            marks = sc.nextInt();
            Integer wrappedMarks = marks;
            list.add(wrappedMarks);
        }

        Integer total = 0;
        Float percent = 0.0f;

        for(Integer a : list){
            total += a;
        }

        percent = total/(float) list.size();

        System.out.println("Marks : " + total + "/" + 500);
        System.out.println("Percentage : " + percent + "%");

    }

}
```

Output:



Conclusion:

Hence, by performing this practical I learnt about the concepts of Wrapper classes and Vector. I also coded, debugged and executed java programs based on the concepts of Wrapper ~~classes~~ classes and Vectors.

Practical No. 06

Aim: Create, debug and run java programs based on classes with objects, method overloading and constructor overloading.

Practical No. 06

Aim: Create, debug and run java programs based on classes with objects, method overloading and constructor overloading.

Theory:

What is a class in Java?

→ i) Classes are the building blocks of programs built using the object oriented methodology.
ii) Objects have certain similar traits - state and behaviour.
iii) The commonality is provided in a blueprint or template for the instantiation of all similar objects.
iv) This blueprint is known as a class.

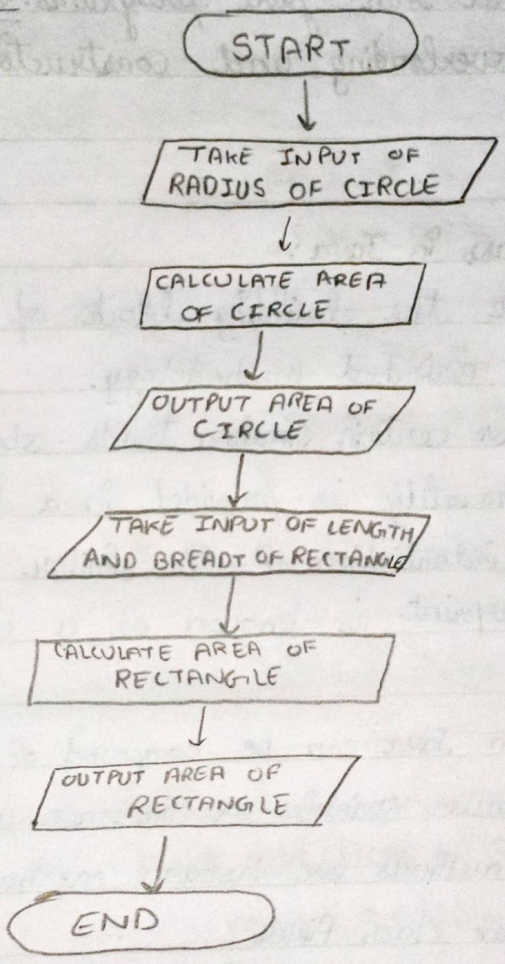
- Every class in Java can be composed of the following elements:
  i) Fields, member variables or instance variables
  ii) member methods or instance methods.
  iii) Static or class fields
  iv) Static or class methods
  v) Inner class.

Method overloading:

If a class has multiple methods having same name but different in parameters, it is known as method overloading.

There are two ways to overload the method in Java:
1) By changing number of arguments
2) By changing the data type

Flow charts:



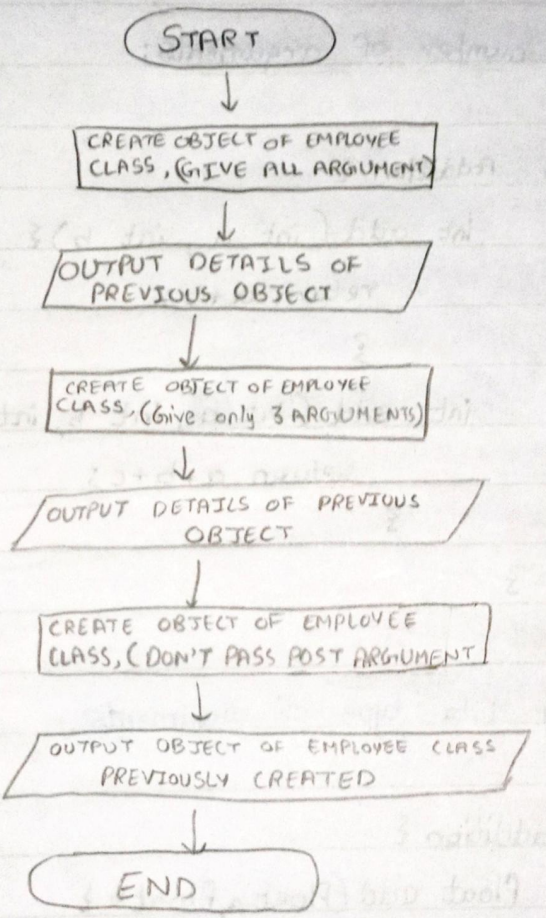
Code 1: Classes, objects, Method Overloading

i) By changing number of arguments:

```
class Addition {
    int add (int a, int b){
        return a+b;
    }
    int add (int a, int b, int c){
        return a+b+c;
    }
}
```

ii) By changing data type of arguments.

```
class addition {
    Float add (Float a, Float b) {
        return a+b;
    }
    int add (int a, int b){
        return a+b;
    }
}
```



Code 2 : Constructor Overloading

Constructor overloading:

Constructor overloading in Java is a technique of having more than one constructor with different parameter lists.
These constructors are differentiated by the compiler by the number of parameters in the list and their types.

Example:

```
class staff {
string Id ;
String name;
int age;

staff ( ) {
    age = 0 ;
    Id = "00000000";
    name = "User-none";
}

staff (String name, Strid Id, & int age ) {
    this.name = name;
    this.id = Id;
    this.age = age;
}
```

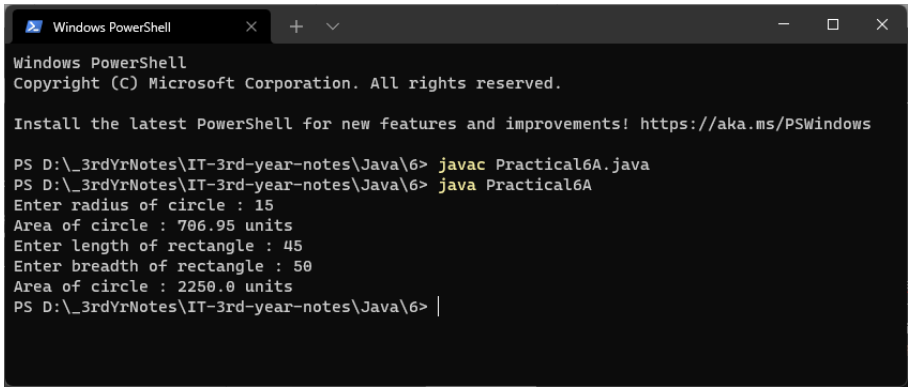
Conclusion:

Hence, by performing this practical I learnt about the various concepts of classes, objects, method overloading and constructor overloading. I also created, debugged and executed java programs based on the above concepts.

Code:

```
import java.util.Scanner;

class calculations{

    final float PI = 3.142f ;

    float area(float radius){
        return radius*radius*PI;
    }

    float area(float length,float breadth){
        return length * breadth;
    }

}

class Practical6A{

    public static void main(String[] args){
        calculations obj =new calculations();
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter radius of circle : ");
        float radius = sc.nextFloat();
        System.out.println("Area of circle : " + obj.area(radius) + " units"
);
        System.out.print("Enter length of rectangle : ");
        float length = sc.nextFloat();
        System.out.print("Enter breadth of rectangle : ");
        float breadth = sc.nextFloat();
        System.out.println("Area of circle : " + obj.area(length,breadth) + "
units" );
    }

}
```

Output:

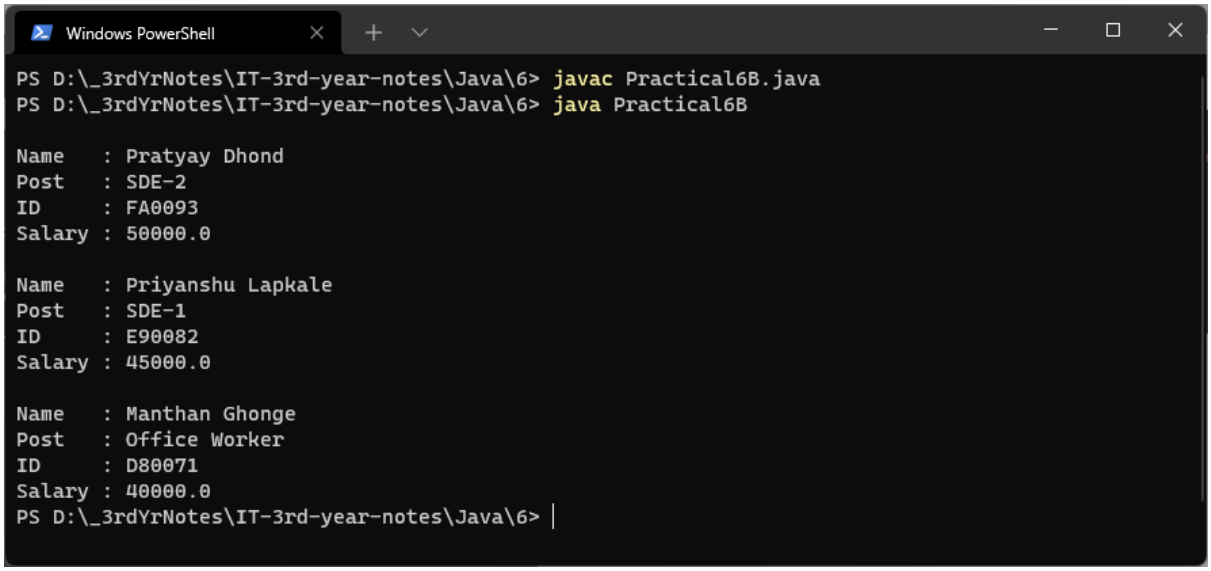


Code:

```
class Employee{
  String name;
  float Salary;
  String position;
  String id;

  Employee(String name, float Salary, String position, String id){
    this.name = name;
    this.Salary = Salary;
    this.position = position;
    this.id = id;
  }

  Employee(String name, float Salary, String id){
    this.name = name;
    this.Salary = Salary;
    this.position = "Office Worker";
    this.id = id;
  }

  Employee(String name, String position, String id){
    this.name = name;
    this.Salary = 45000;
    this.position = position;
    this.id = id;
  }

  void display(){
    System.out.println();
    System.out.println("Name   : " + name);
    System.out.println("Post   : " + position);
    System.out.println("ID     : " + id);
    System.out.println("Salary : " + Salary);
  }

}

class Practical6B{

  public static void main(String[] args){
      Employee e1 = new Employee("Pratyay Dhond",50000,"SDE-2","FA0093");
      e1.display();

      Employee e2 = new Employee("Priyanshu Lapkale","SDE-1","E90082");
      e2.display();
```

```
        Employee e3 = new Employee("Manthan Ghonge",40000,"D80071");
        e3.display();


    }

}
```

Output:



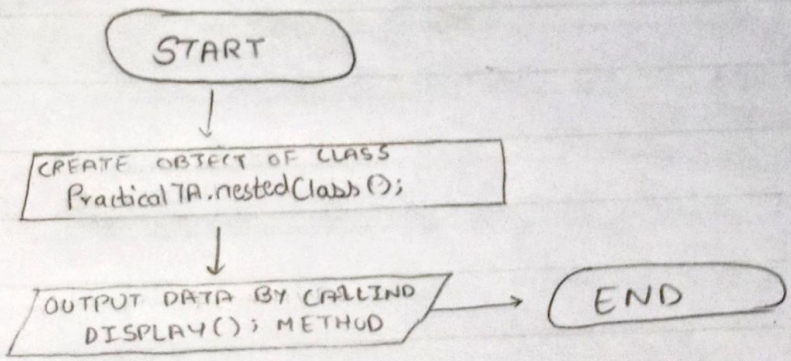
Conclusion:

Hence, by performing this practical I learnt about the various concepts of classes, objects, method overloading and constructor overloading. I also created, debugged and executed java programs based on the above concepts.

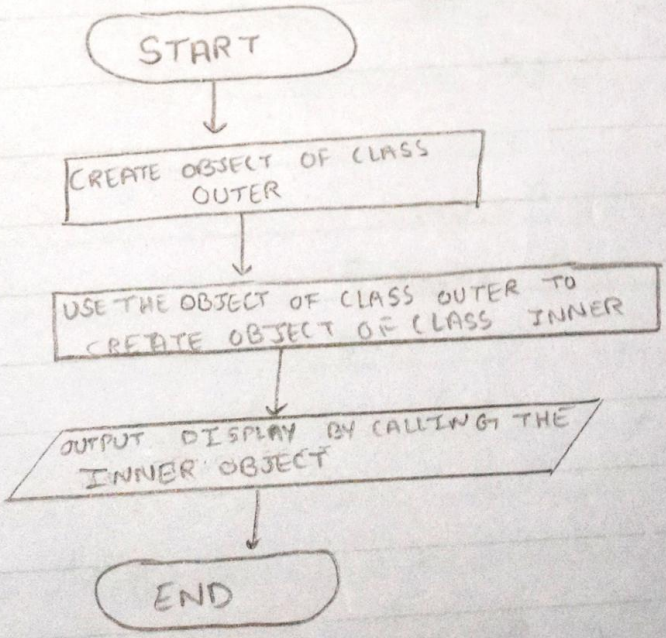
Practical No. 07

Aim : Create, debug and run java programs based on Nested and Inner classes

Flow chart :



Code 1: Nested Class



Code 2: Inner classes

Practical No. 07

Aim: Create, debug and run java programs based on Nested and Inner classes.

Theory:

What is a nested class?

→ • A class which is defined within another class is called as nested class.

• The scope of the nested class is bounded by the scope of the ~~inner classes~~ enclosing classes.

• Example:

```
public class School {
    class student {
        st String name;
        student(){
            name = "User";
        } marks = 0;
    }
    private int marks;
}
School(){

    }
}
```

What is an Inner class?

→ • An inner class is nested but non-static and it is the most important of all nested classes.

• It has access to all the members of the outer classes

• Example,

```
class Outer {
    int x = 5;
    class Inner {
        int y;
    }
}

class Main {

    public static void main(String[] args){
        Outer out = new Outer();
        Outer.Inner Inn = out.new Inner();
        System.out.println( out.x + Inn.y);
    }
```

Conclusion:

Hence, I learnt the concepts of Inner classes and nested classes and thus created, developed and executed java programs based on Nested and Inner classes.

Code:

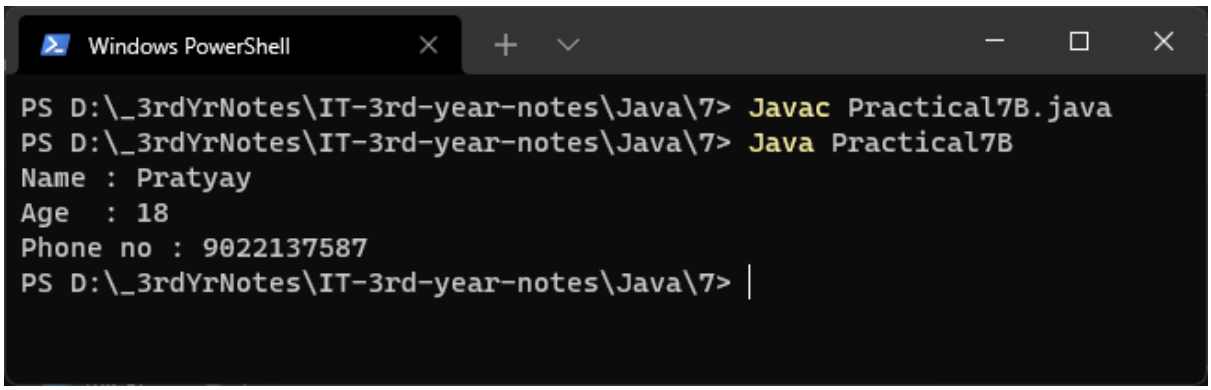
```java
//nested class


class Practical7A{

    static int age = 18;
    String phoneNo = "9022137587";
    static private String name = "Pratyay";

    static class nestedClass{
      void display(){
        System.out.println("Name : " + name);
        System.out.println("Age  : " + age);

        // Can't access non static variables
        // System.out.println("Phone no : " + phoneNo);
      }

    }


  public static void main(String[] args){
    Practical7A.nestedClass nestedObject = new Practical7A.nestedClass();
    nestedObject.display();

  }


}
```

Output:

Code:

```java
//nested class

class Outer{

    static int age = 18;
    String phoneNo = "9022137587";
    static private String name = "Pratyay";

    class innerClass{
      void display(){
        System.out.println("Name : " + name);
        System.out.println("Age  : " + age);
        System.out.println("Phone no : " + phoneNo);
      }

    }

}

class Practical7B{

  public static void main(String[] args){
    Outer outerObj = new Outer();
    Outer.innerClass innerObj = outerObj.new innerClass();
    innerObj.display();


  }


}
```

Output:



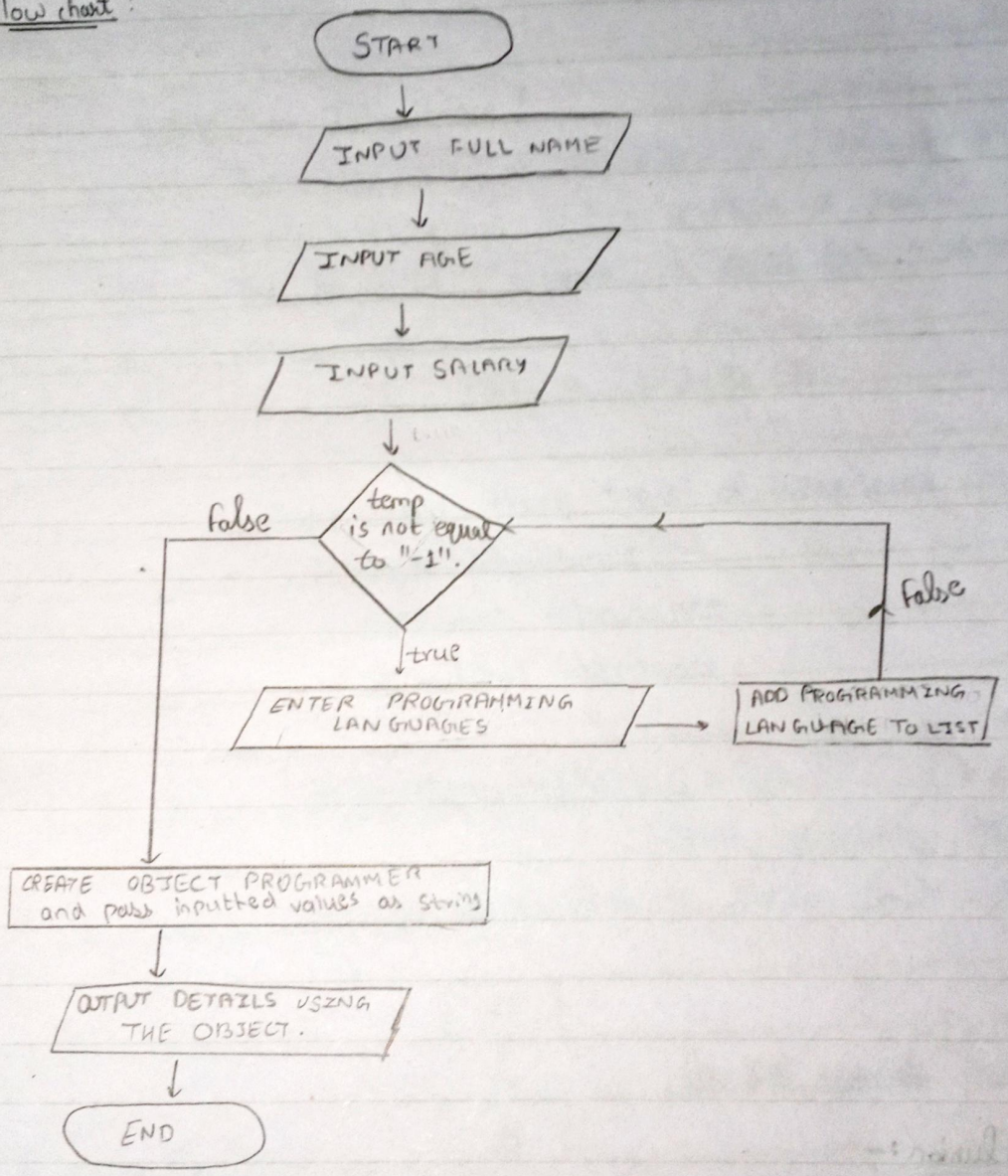
Conclusion:

Hence, I learnt the concepts of Inner classes and nested classes and thus created, developed, and executed java programs based on Nested and Inner classes.

Practical No. 08

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

Flow chart:



Code 1: SINGLE INHERITANCE

Practical No. 08

Aim: Create debug and run Java programs based on Inheritance.

Theory:

What is Inheritance?

→ The capability of a class to derive properties and characteristics from another class is called Inheritance. The keyword 'extends' is used for inheritance java.

For example:

```
class A extends B;
```

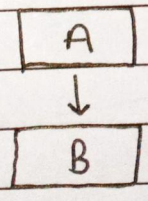
There are three types of inheritance in java.

i) Single inheritance
ii) Multi-level inheritance
iii) Hierachical inheritance

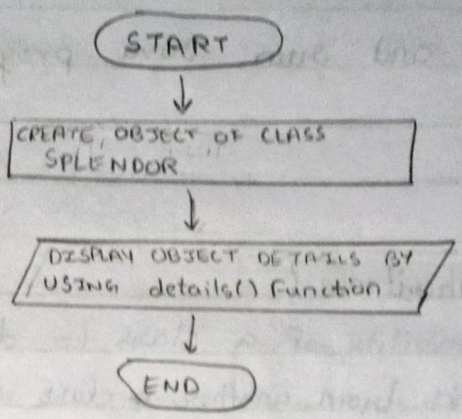
i) Single Inheritance:

When a class ~~extends~~ inherits another class, it is known as a single inheritance.

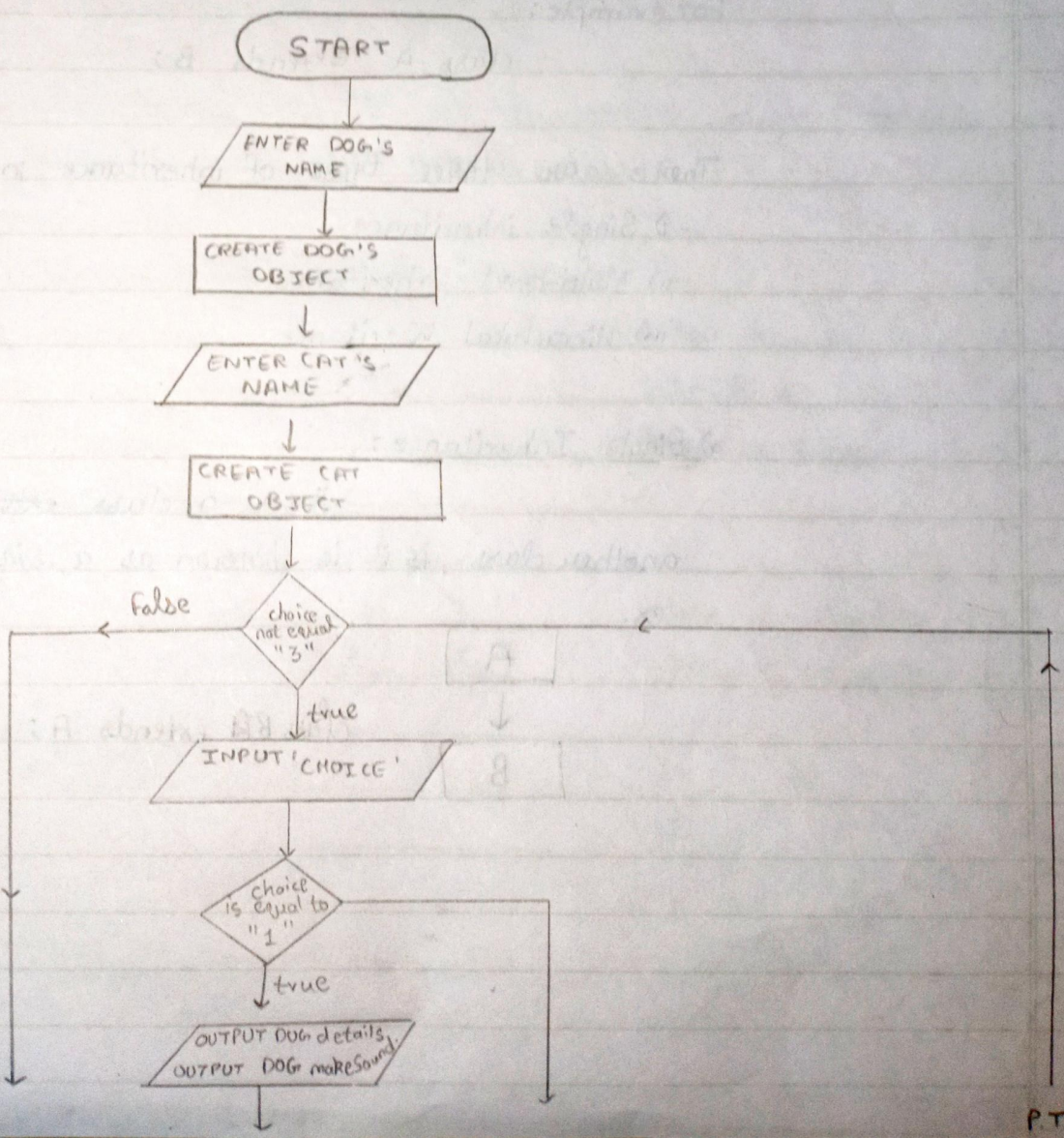
ex.



class B extends A;



CODE 2 : MULTI-LEVEL INHERITANCE

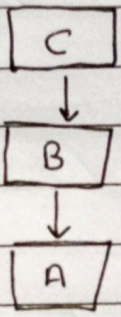


PTO

ii) Multi level Inheritance:

When their is a chain of inheritance. It is known as Multi-level Inheritance.

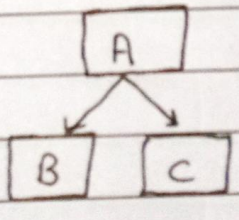
Example:



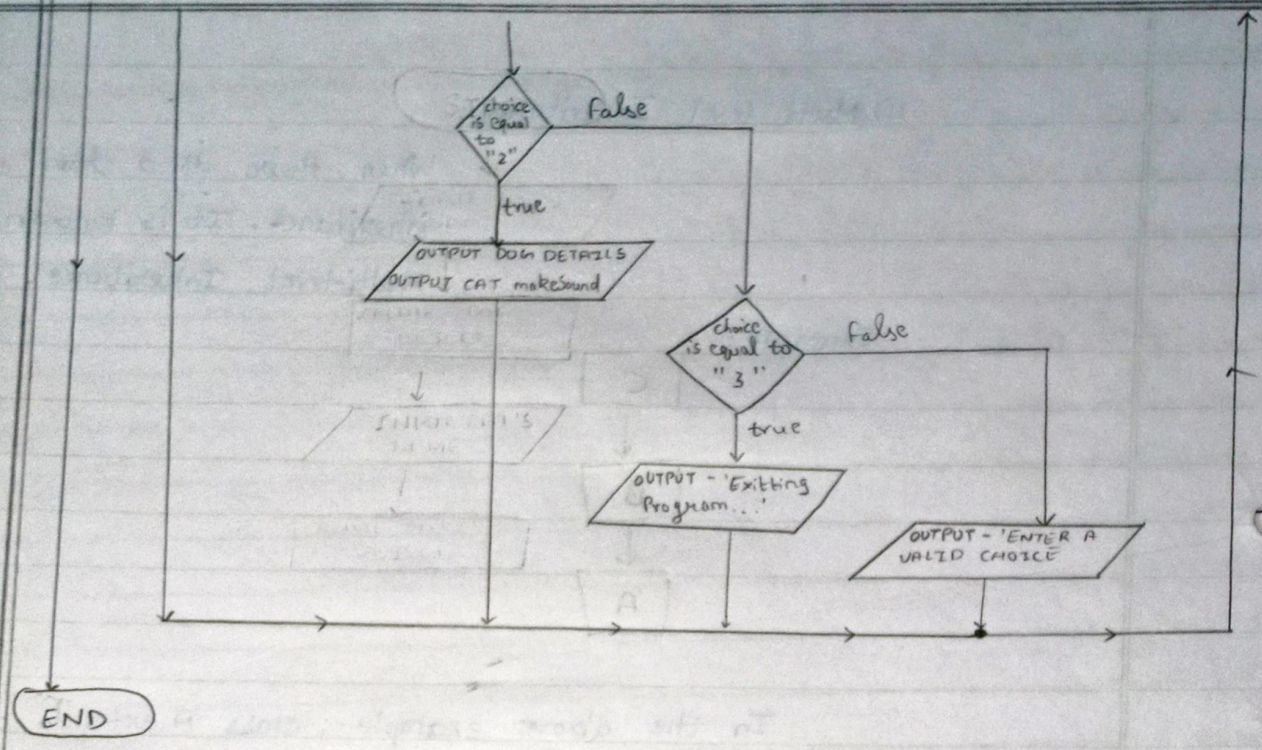
In the above example, class A extends class B which extends class C.

iii) Hierarchical Inheritance:

When two or more classes inherit a single class, it is known as hierarchical inheritance.



class B extends A;
class C extends A;



Code 3 : Hierarchical Inheritance.

Conclusion:

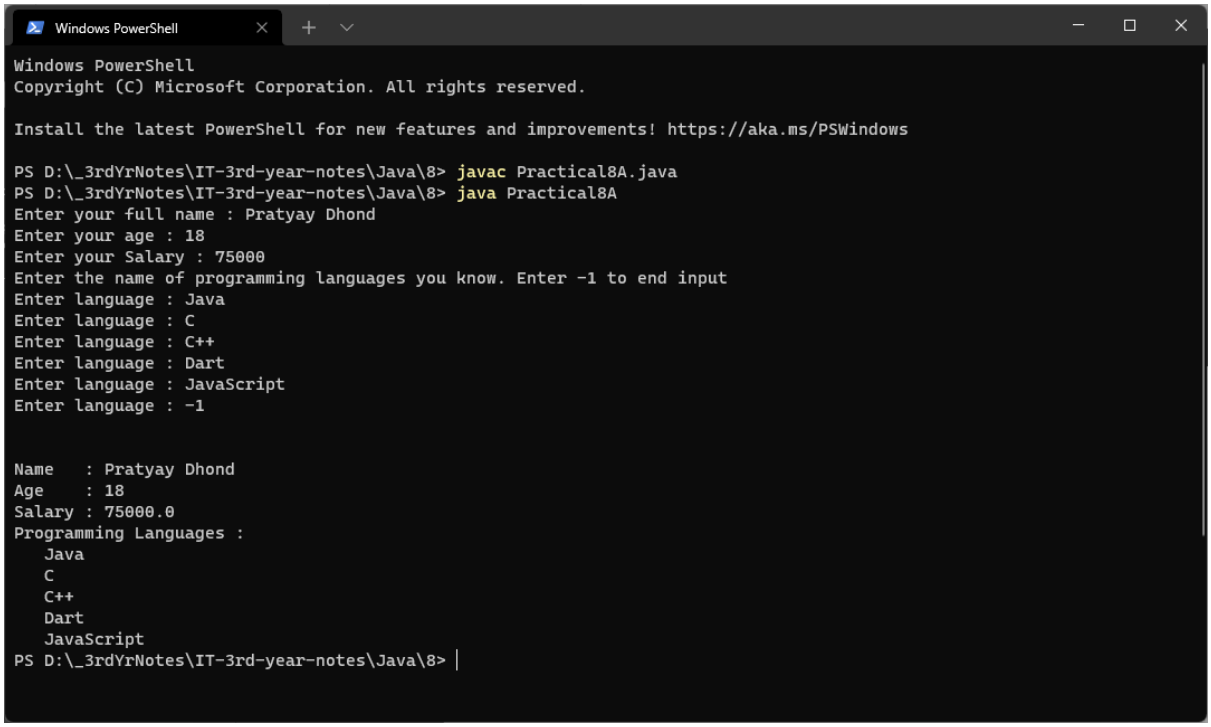
Hence, by performing this practical I learnt about the concepts of inheritance. I also created, debugged and executed Java programs on these concepts.

Code:

```java
// single inheritance

import java.util.Scanner;
import java.util.Vector;

class Employee{
    String name;
    int age;
    float salary;

    void display(){
        System.out.println("Name   : " + name);
        System.out.println("Age    : " + age);
        System.out.println("Salary : " + salary);
    }

    Employee(String name,int age,float salary){
        this.name = name;
        this.age = age;
        this.salary = salary;
    }

}

class Programmer extends Employee{

    String[] languages;

    Programmer(String name, int age, float salary, String[] languages){
        super(name,age,salary);
        this.languages = languages;
    }


    void display(){
        System.out.println("\n");
        super.display();
        System.out.println("Programming Languages : ");
        for(String a : languages){
            System.out.println("    " + a);
        }
    }

}

class Practical8A{
```

```
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String name;
        int age;
        float salary;
        Vector<String> list = new Vector<>();

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

        System.out.print("Enter your age : ");
        age = sc.nextInt();

        System.out.print("Enter your Salary : ");
        salary = sc.nextFloat();
        sc.nextLine();
        System.out.println("Enter the name of programming languages you know.
Enter -1 to end input");

        String temp;
        do{
            System.out.print("Enter language : ");
            temp = sc.nextLine();
            if(!temp.equals("-1") )
                list.add(temp);
        }while(!temp.equals("-1"));

        String[] languages = new String[list.size()];
        for(int i = 0; i < list.size(); i++)
            languages[i] = list.elementAt(i);

        Programmer p1 = new Programmer(name,age,salary,languages);
        p1.display();
    }

}
```

Output:

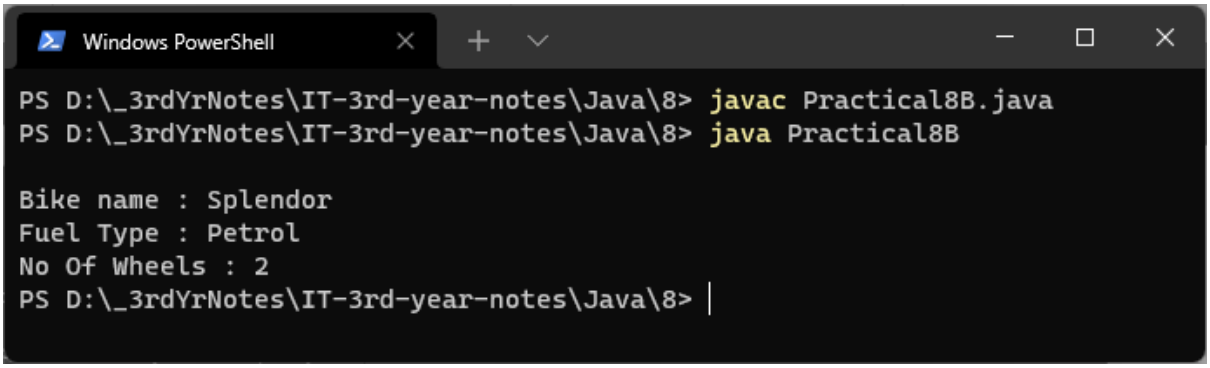

Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\8> javac Practical8A.java
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\8> java Practical8A
Enter your full name : Pratyay Dhond
Enter your age : 18
Enter your Salary : 75000
Enter the name of programming languages you know. Enter -1 to end input
Enter language : Java
Enter language : C
Enter language : C++
Enter language : Dart
Enter language : JavaScript
Enter language : -1
Name : Pratyay Dhond
Age : 18
Salary : 75000.0
Programming Languages :
Java
C
C++
Dart
JavaScript
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\8>

Code:

```
      // multi level inheritance


import java.util.Scanner;
import java.util.Vector;

class Vehicle{
    String fuelType;

    Vehicle(String fuelType){
        this.fuelType = fuelType;
    }

    String getFuelType(){
        return fuelType;
    }
}

class Bike extends Vehicle{
    int noOfWheels;

    Bike(String fuelType,int noOfWheels){
        super(fuelType);
        this.noOfWheels = noOfWheels;
    }

    void displayFuelType(){
        System.out.println("Fuel Type : " + super.getFuelType());
    }

    void displayNoOfWheels(){
        System.out.println("No Of Wheels : " + noOfWheels);
    }
}

class Splendor extends Bike{
    String modelName = "Splendor";

    Splendor(String fuelType,int noOfWheels){
        super(fuelType,noOfWheels);
    }

    void display(){
        System.out.println("\nBike name : " + modelName);
        super.displayFuelType();
        super.displayNoOfWheels();
    }
}
```

```java
class Practical8B{

    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        String fuelType = "Petrol";
        int noOfWheels = 2;

        Splendor b1 = new Splendor(fuelType,noOfWheels);
        b1.display();

    }

}
```

Output:

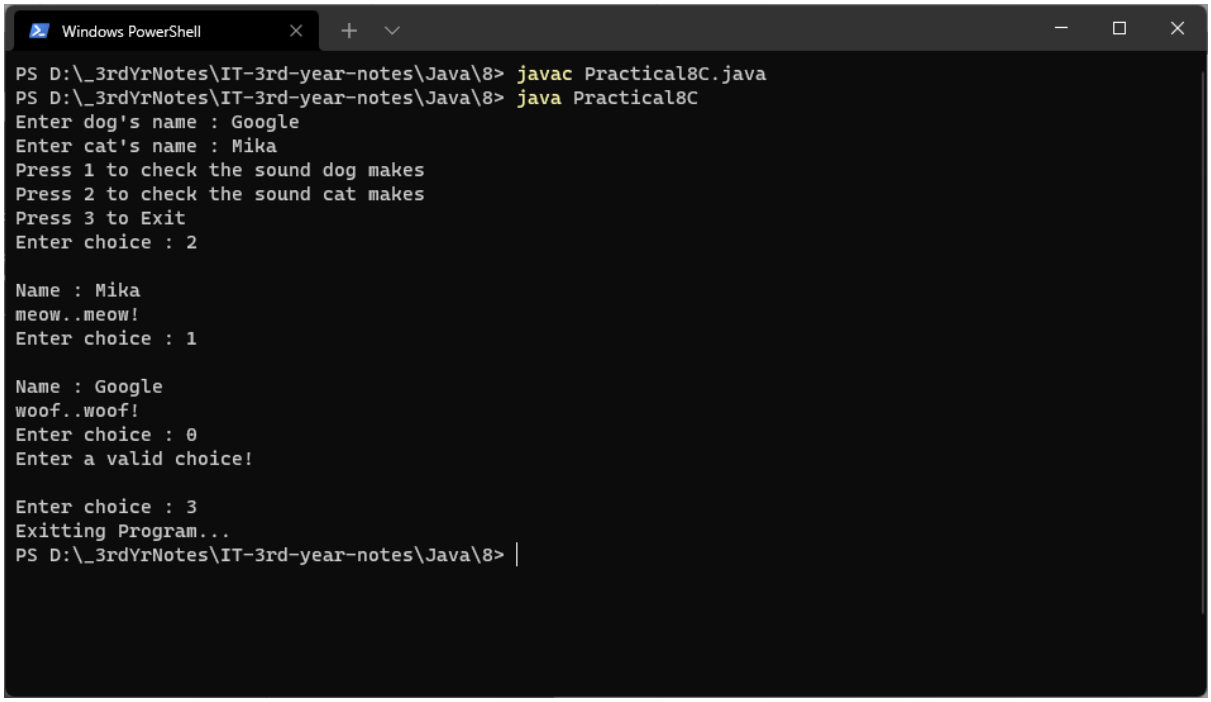


Code:

```
// hierachical inheritance


import java.util.Scanner;

abstract class Animal{
    String name;

    Animal(String name){
        this.name = name;
    }

    String getName(){
        return name;
    }

    abstract void makeSound();

}

class dog extends Animal{

    dog(String name){
        super(name);
    }

    void details(){
        System.out.println("\nName : " + super.getName());
    }

    void makeSound(){
        System.out.println("woof..woof!");
    }

}

class cat extends Animal{

    cat(String name){
        super(name);
    }

    void details(){
        System.out.println("\nName : " + super.getName());
    }

    void makeSound(){
        System.out.println("meow..meow!");
```

```
    }

}

class Practical8C{

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

        String name;

        System.out.print("Enter dog's name : ");
        name = sc.next();
        dog d1 = new dog(name);

        System.out.print("Enter cat's name : ");
        name = sc.next();
        cat c1 = new cat(name);

        System.out.println("Press 1 to check the sound dog makes\nPress 2 to
check the sound cat makes\nPress 3 to Exit");
        String choice;

        do{
            System.out.print("Enter choice : ");
            choice = sc.next();

            if(choice.equals("1")){
                d1.details();
                d1.makeSound();
            }else if(choice.equals("2")){
                c1.details();
                c1.makeSound();
            }else if(choice.equals("3")){
                System.out.println("Exitting Program...");
            }else{
                System.out.println("Enter a valid choice! \n");
            }
        }while(!choice.equals("3"));

    }

}
```

Output:


Windows PowerShell
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\8> javac Practical8C.java
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\8> java Practical8C
Enter dog's name : Google
Enter cat's name : Mika
Press 1 to check the sound dog makes
Press 2 to check the sound cat makes
Press 3 to Exit
Enter choice : 2
Name : Mika
meow..meow!
Enter choice : 1
Name : Google
woof..woof!
Enter choice : 0
Enter a valid choice!
Enter choice : 3
Exitting Program...
PS D:_3rdYrNotes\IT-3rd-year-notes\Java\8>

Conclusion:

Hence, by performing this practical I ~~the~~ learnt about the concepts of inheritance. I also created, debugged and ~~exe~~ executed Java programs based on these concepts.