LAB MANUAL
for
OOPS LAB USING JAVA
Paper Code: (CC -12)

5th Semester

B.Sc. in Computer Science (Hons.)

<u>**Prepared by**</u>

RANU CHOWDHURY
Department of Comp. Sc.

SIR GURUDAS MAHAVIDYALAYA

1) Write a program in Java that sorts half of the elements in ascending and the rest half of the elements in descending order.

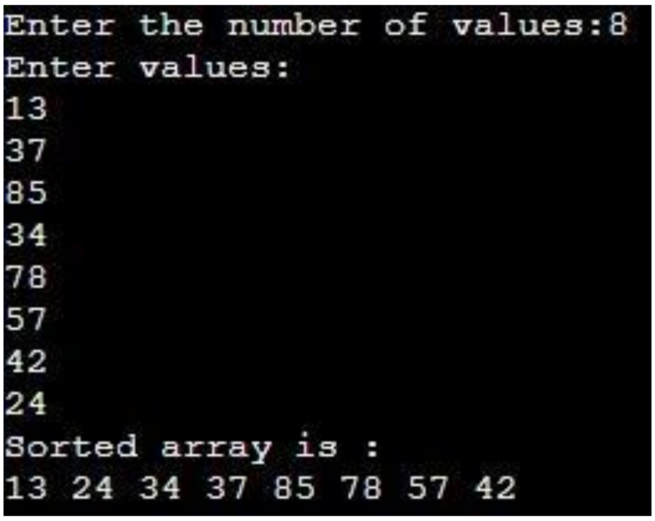
<u>Algorithm:-</u>

```
1. Start
2. Initialize an integer array arr.
3. Prompt the user to enter the number of values n.
4. Create a Scanner object in to read input from the user.
5. Initialize a loop to read n integer values from the user and store them in the
array arr:
i) Loop from i = 0 to n - 1:
ii) Read an integer from the user and store it in arr[i].
6. Sort the array arr in ascending order using Arrays.sort(arr).
7. Print "Sorted array is :" to the console.
8. Initialize a loop to print the first half of the sorted array in ascending order:
 i) Loop from i = 0 to n / 2 - 1:
ii) Print arr[i] followed by a space.
9. Initialize a loop to print the second half of the sorted array in descending
order:
i) Loop from j = n - 1 down to n / 2:
ii) Print arr[j] followed by a space.
10. Close the Scanner object in.
11. End
```

<u>Source Code:-</u>

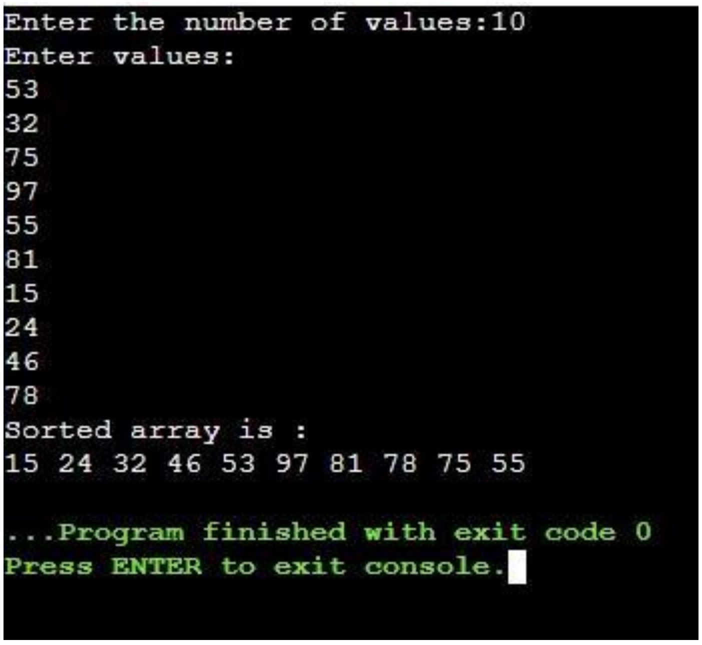
```
import java.util.*;
class Q1{
 public static void printOrder(int[] arr, int n) {
 Arrays.sort(arr);
 for(int i = 0; i <n / 2; i++)
 System.out.print(arr[i] + " ");
 for(int j = n - 1; j >= n / 2; j--)
 System.out.print(arr[j] + " ");
 }
 public static void main(String[] args){
 Scanner in = new Scanner(System.in);
 System.out.print("Enter the number of values:");
 int n = in.nextInt();
 int[] arr = new int[n];
 System.out.println("Enter values:");
 for(int i =0; i < n; i++)
 arr[i] = in.nextInt();
 System.out.println("Sorted array is :");
 printOrder(arr, n);
 in.close();
 }
}
```

Output 1 :-



```
Enter the number of values:8
Enter values:
13
37
85
34
78
57
42
24
Sorted array is :
13 24 34 37 85 78 57 42
```

Output 2 :-



```
Enter the number of values:10
Enter values:
53
32
75
97
55
81
15
24
46
78
Sorted array is :
15 24 32 46 53 97 81 78 75 55

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

2) Write a program in java that accepts a 2D matrix and prints the matrix with row minimum and column minimum values.

```
4 3 5  3
1 0 7  0
8 4 6  4
1 0 5
```

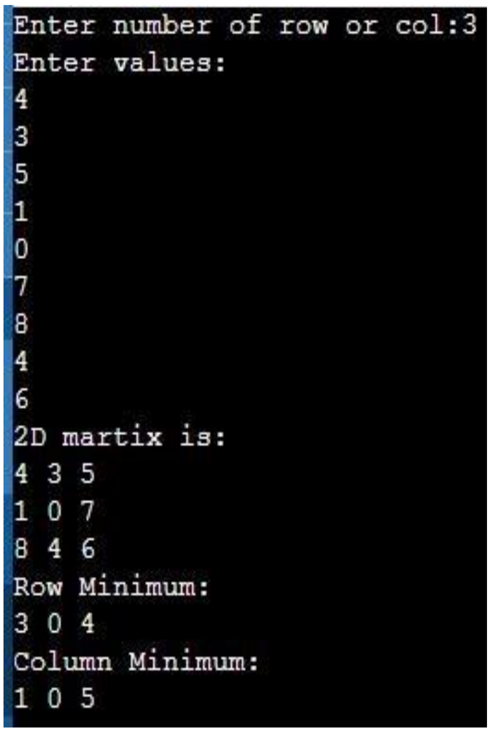
Algorithm:-

```
1. Start
2. Create a Scanner object in to read input from the user.
3. Declare integer variables n, rowmin, and colmin.
4. Prompt the user to enter the number of rows or columns, n.
5. Initialize a 2D integer array arr of size n x n.
6. Initialize integer arrays row and col, both of size n, to store the minimum
values in rows and columns.
7. Print "Enter values:" to indicate that the user should input the elements of
the 2D matrix.
8. Create nested loops to read n x n integer values from the user and store
them in the arr matrix:
i) Loop from i = 0 to n - 1:
ii) Loop from j = 0 to n - 1:
iii) Read an integer from the user and store it in arr[i][j].
9. Print "2D matrix is:" to display the entered matrix.
10.Create nested loops to iterate over each element of the arr matrix and find
the minimum values in rows and columns:
i) Loop from i = 0 to n - 1:
ii) Initialize rowmin to a large value to find the minimum value in the
current row.
iii) Initialize colmin to a large value to find the minimum value in the
current column.
iv) Loop from j = 0 to n - 1:
   a) If arr[i][j] is less than rowmin, update rowmin to arr[i][j].
   b) If arr[j][i] is less than colmin, update colmin to arr[j][i].
v) Store rowmin in the row array at index i.
vi) Store colmin in the col array at index i.
11.Print "Row Minimum:" to indicate the minimum values in each row.
12.Loop through the row array and print each minimum value for rows.
13.Print a newline character to separate the row and column minimum values.
14.Print "Column Minimum:" to indicate the minimum values in each column.
15.Loop through the col array and print each minimum value for columns.
16.Close the Scanner object in.
17.End
```

Source Code:-

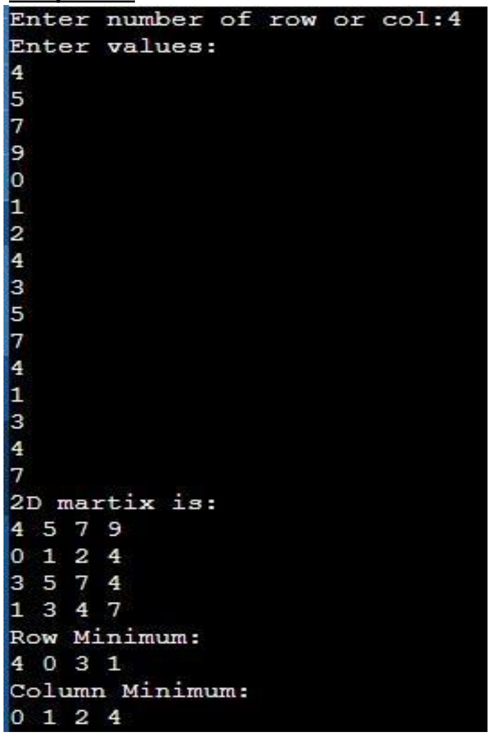
```java
import java.util.*;
public class matrix_q2 {
 public static void main(String[] args) {
 Scanner in = new Scanner(System.in);
 int n;
 int rowmin, colmin;
 System.out.print("Enter number of row or col:");
 n = in.nextInt();
 int[][] arr = new int[n][n];
 int[] row = new int[n];
 int[] col = new int[n];
 System.out.println("Enter values:");
 for(int i =0; i < n; i++){
 for(int j = 0; j < n; j++){
 arr[i][j] = in.nextInt();
}
 }
 System.out.println("2D martix is:");
 for(int i = 0; i< n; i++){
 for(int j = 0; j< n; j++){
 System.out.print(arr[i][j] + " ");
 }
 System.out.println();
 }
 for(int i = 0; i < n; i++){
 rowmin = 999;
 colmin = 999;
 for(int j = 0; j < n; j++){
 if(arr[i][j] < rowmin)
 rowmin = arr[i][j];
 if(arr[j][i] < colmin)
 colmin = arr[j][i];
 }
 row[i] = rowmin;
 col[i] = colmin;
 }
 System.out.println("Row Minimum:");
 for(int i =0; i < n; i++){
 System.out.print(row[i] + " ");
 }
 System.out.println("");
 System.out.println("Column Minimum:");
 for(int i =0; i < n; i++){
 System.out.print(col[i] + " ");
 }
 in.close();
 }
}
```

Output 1:-



```
Enter number of row or col:3
Enter values:
4
3
5
1
0
7
8
4
6
2D martix is:
4 3 5
1 0 7
8 4 6
Row Minimum:
3 0 4
Column Minimum:
1 0 5
```

Output 2:-



```
Enter number of row or col:4
Enter values:
4
5
7
9
0
1
2
4
3
5
7
4
1
3
4
7
2D martix is:
4 5 7 9
0 1 2 4
3 5 7 4
1 3 4 7
Row Minimum:
4 0 3 1
Column Minimum:
0 1 2 4
```

3) Write a program in java to delete all consonants from an input string and print the result string.

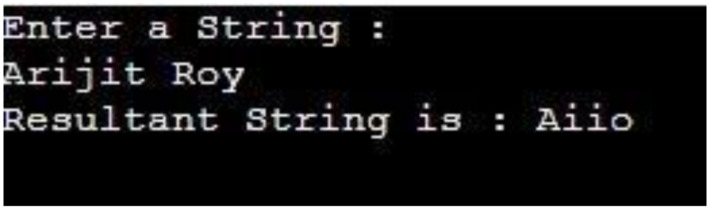
Algorithm :-

```
(i) Start
(ii) Enter a string to store the result.
     (a) Iterate over the input string, and for each character:
     (b) If the character is a consonant, skip it.
(v) Otherwise, add the character to the result string.
(vi) Print the result string.
(vii) Exit the program .
```

Source Code:-

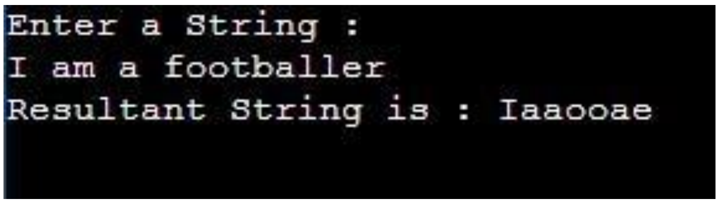
```
import java.util.Scanner;
public class DeleteConsonants {
public static String delConsonant(String str){
StringBuilder ans = new StringBuilder("");
for(int i = 0; i < str.length(); i++){
char ch = str.charAt(i);
if(ch == 'A' || ch == 'a' || ch == 'E' || ch == 'e' || ch == 'I' || ch == 'i' || ch == 'O' || ch == 'o' || ch == 'U' ||
ch == 'u'){
ans.append(ch);
 }
 }
 return ans.toString();
 }
 public static void main(String[] args) {
 Scanner sc = new Scanner(System.in);
 String str = "";
 System.out.println("Enter a String :");
 str = sc.nextLine();
 String res = delConsonant(str);
 System.out.println("Resultant String is : " + res);
 }
 }
```

Output 1



```
Enter a String :
Arijit Roy
Resultant String is : Aiio
```

Output 2



```
Enter a String :
I am a footballer
Resultant String is : Iaaooae
```

Q4} A class called MyPoint, which models a 2D point with x and y coordinates. It contains:
• Two instance variables x (int) and y (int).
• A default (or "no-argument" or "no-arg") constructor that construct a point at the default location of (0, 0).
• A overloaded constructor that constructs a point with the given x and y coordinates.
• A method setXY() to set both x and y.
• A method getXY() which returns the x and y in a 2-element int array.
• A toString() method that returns a string description of the instance in the format "(x, y)".
• A method called distance(int x, int y) that returns the distance from this point to another point at the given (x, y) coordinates, Write the MyPoint class. Also write a test driver (called TestMyPoint) to test all the public methods defined in the class.

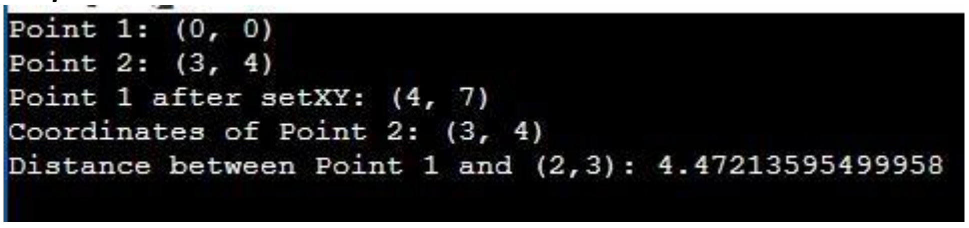
Algorithm:-

1) Create a class named MyPoint.
2) Declare two private instance variables, x and y, to store the coordinates of the point.
3) Write a default constructor (no-argument constructor) that initializes the point to the default location (0, 0).
4) Write an overloaded constructor that takes two parameters (x and y) and initializes the point to the specified coordinates.
5) Implement a method setXY(int x, int y) to set the values of both x and y.
6) Implement a method getXY() that returns a 2-element int array containing the values of x and y.
7) Implement a method toString() that returns a string representation of the point in the format "(x, y)".
8) Implement a method distance(int otherX, int otherY) that calculates and returns the distance between the current point and another point with coordinates (otherX, otherY).
9) In the main program or test driver:
   a. Create instances of MyPoint using both the default and overloaded constructors.
   b. Use setXY to change the coordinates of a point.
   c. Call getXY to retrieve the coordinates and print them.
   d. Calculate and print the distance between two points using the distance method.
   e. Print the string representation of a point using the toString method.
10) Run the program and verify that the output matches the expected results.

Source Code:-

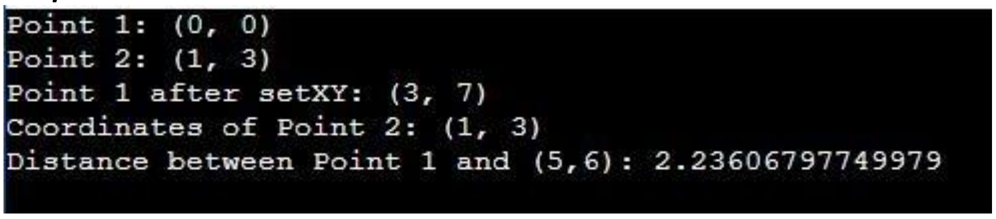
```
class MyPoint {
 private int x;
 private int y;
 // Default constructor
 public MyPoint() {
 this.x = 0;
 this.y = 0;
 }
 // Overloaded constructor
 public MyPoint(int x, int y) {
 this.x = x;
 this.y = y;
 }
 // Set both x and y
 public void setXY(int x, int y) {
 this.x = x;
 this.y = y;
 }
 // Get x and y in a 2-element int array
 public int[] getXY() {
 int[] coordinates = {x, y};
 return coordinates;
 }
 // Calculate distance from this point to another point
 public double distance(int otherX, int otherY) {
 int dx = this.x - otherX;
 int dy = this.y - otherY;
 return Math.sqrt(dx * dx + dy * dy);
 }
 // String description of the instance
 @Override
 public String toString() {
 return "(" + x + ", " + y + ")";
 }
}
public class TestMyPoint {
 public static void main(String[] args) {
 // Test default constructor
 MyPoint point1 = new MyPoint();
 System.out.println("Point 1: " + point1);
 // Test overloaded constructor
 MyPoint point2 = new MyPoint(3, 4);
 System.out.println("Point 2: " + point2);
 // Test setXY method
 point1.setXY(4, 7);
 System.out.println("Point 1 after setXY: " + point1);
 // Test getXY method
 int[] coordinates = point2.getXY();
 System.out.println("Coordinates of Point 2: (" + coordinates[0] + ", " + coordinates[1] + ")");
 // Test distance method
 double distance = point1.distance(2, 3);
 System.out.println("Distance between Point 1 and (2,3): " + distance);
 }
}
```

Output 1:-



```
Point 1: (0, 0)
Point 2: (3, 4)
Point 1 after setXY: (4, 7)
Coordinates of Point 2: (3, 4)
Distance between Point 1 and (2,3): 4.47213595499958
```

Output 2:-



```
Point 1: (0, 0)
Point 2: (1, 3)
Point 1 after setXY: (3, 7)
Coordinates of Point 2: (1, 3)
Distance between Point 1 and (5,6): 2.23606797749979
```

Q5} Create a superclass 'Person' and two subclasses 'Student' and 'Staff'. The following are the instance variables and methods:
a. For 'Person' instance variables: name:String, address:String. Initiate variable through constructor, incorporate one method setPerson() that updates Person variables , another method tostring() that shows Person details as "Person[name=?,address=?".
b. For 'Student' sub class instance variables: program:String, year:String, fees:double. Initiate both 'Student' and 'Person' variables through constructor, incorporate one method setStudent() that updates both student and 'Person' data, another method tostring() that shows 'Person-Student' details as "Person[name=?,address=?,Program=?,Year=?,Fees=?".
c. For 'Staff' subclass instance variables: school:String, pay:double. Initiate both 'Staff' and 'Person' variables through constructor, incorporate one method setStaff() that updates both 'staff' and 'Person' data, another method tostring() that shows 'Person-Staff' details as "Person[name=?,address=?,School=?,Pays=?".
Write the classes and a test driver main class to test all functions mentioned above.

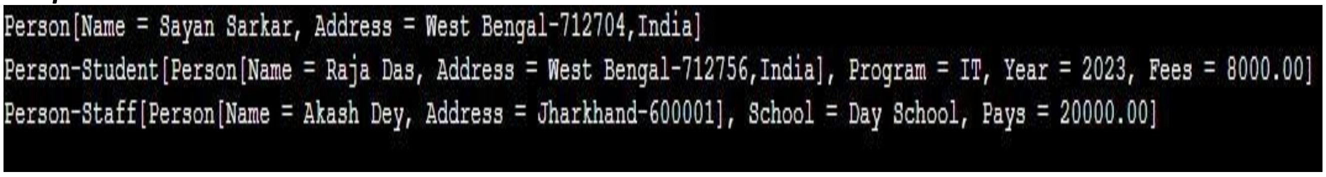
Algorithm :-

```
1) Initialise name and address with values provided as parameters.
2) Set the name and address attributes with the provided values.
3) Return a formatted string representing the person's information.
4) Call the superclass (Person) constructor to initialise name and address.
    i) Initialize program, year, and fees with provided values.
5) Call the setPerson method of the superclass to set name and address.
    i)Set program, year, and fees with provided values.
6) Override toString() from the superclass.
    i) Return a formatted string representing the person-student's information.
7) Call the superclass (Person) constructor to initialize name and address.
    i) Initialise school and pay with provided values.
8) Call the setPerson method of the superclass to set name and address.
    i) Set school and pay with provided values.
9) Override toString() from the superclass.
    i) Return a formatted string representing the person-staff's information.
10) In the main method :
    i) Create an instance of the Person class (testPerson) with sample data and print its information.
    ii) Create an instance of the Student class (testStudent) with sample data and print its information.
    iii) Create an instance of the Staff class (testStaff) with sample data and print its information.
```

Source Code :-

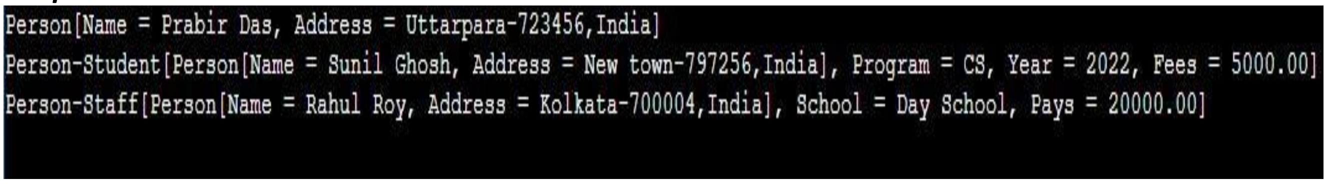
```
class Person  {
 protected String name;
 protected String address;
 public Person(String name, String address) {
 this.name = name;
 this.address = address;
 }
 public void setPerson(String name, String address) {
 this.name = name;
 this.address = address;
 }
 @Override
 public String toString() {
 return String.format("Person[Name = %s, Address = %s]", name, address);
 }
}
class Student extends Person {
 private String program;
 private String year;
 private double fees;
 public Student(String name, String address, String program, String year, double fees) {
 super(name, address);
 this.program = program;
 this.year = year;
 this.fees = fees;
 }
 public void setStudent(String name, String address, String program, String year, double fees) {
 super.setPerson(name, address);
 this.program = program;
 this.year = year;
 this.fees = fees;
 }
 @Override
 public String toString() {
 return String.format("Person-Student[%s, Program = %s, Year = %s, Fees = %.2f]", super.toString(),
program, year, fees);
 }
}
class Staff extends Person {
 private String school;
 private double pay;
 public Staff(String name, String address, String school, double pay) {
 super(name, address);
 this.school = school;
 this.pay = pay;
 }
 public void setStaff(String name, String address, String school, double pay) {
 super.setPerson(name, address);
 this.school = school;
 this.pay = pay;
 }
 @Override
 public String toString() {
 return String.format("Person-Staff[%s, School = %s, Pays = %.2f]", super.toString(), school, pay);
 }
}
// Test driver main class
public class TestPerson {
 public static void main(String[] args) {
 Person testPerson = new Person("Sayan Sarkar", "West Bengal-712704,India");
 System.out.println(testPerson.toString());
 Student testStudent = new Student("Raja Das", "West Bengal-712756,India", "IT", "2023",8000);
 System.out.println(testStudent.toString());
 Staff testStaff = new Staff("Akash Dey", "Jharkhand-600001", "Day School", 20000);
 System.out.println(testStaff.toString());
 }
}
```

Output 1:-



```
Person[Name = Sayan Sarkar, Address = West Bengal-712704,India]
Person-Student[Person[Name = Raja Das, Address = West Bengal-712756,India], Program = IT, Year = 2023, Fees = 8000.00]
Person-Staff[Person[Name = Akash Dey, Address = Jharkhand-600001], School = Day School, Pays = 20000.00]
```

Output 2:-



```
Person[Name = Prabir Das, Address = Uttarpara-723456,India]
Person-Student[Person[Name = Sunil Ghosh, Address = New town-797256,India], Program = CS, Year = 2022, Fees = 5000.00]
Person-Staff[Person[Name = Rahul Roy, Address = Kolkata-700004,India], School = Day School, Pays = 20000.00]
```

Q6} Create a base class 'Square' having instance variable side:double. Initiate variable using constructor, a method 'getVolume() : double' that calculates volume and print it. Create a derived class 'Cylinder' having instance variable height:double. Initiate variables of both classes through constructor, override method 'getVolume() : double' to calculate volume of cylinder taking 'side' variable of base class as 'radius' and print it.

Algorithm :-

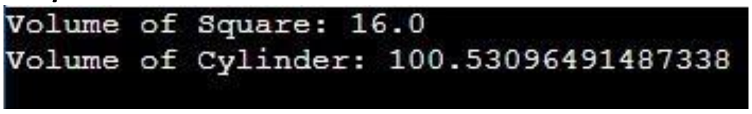
```
1) Create Square Class:
    a. Make a class called Square.
    b. Inside the class, add a variable called side to represent the length of a side of the square.
2) Initialize Square's Side:
    c. Create a constructor for the Square class to set the value of the side variable.
3) Calculate Square Volume:
    d. Add a method in the Square class called getVolume that calculates the volume of the square
       (side squared) and prints it.
4) Create Cylinder Class (Derived from Square):
    a. Make another class called Cylinder that extends or inherits from the Square class.
    b. Add a new variable to the Cylinder class called height to represent the height of the cylinder.
5) Initialize Cylinder's Variables:
    a. Create a constructor for the Cylinder class to set the values of both the side (inherited from Square)
       and the new height variable.
6) Override getVolume Method for Cylinder:
    a. Override the getVolume method in the Cylinder class to calculate the volume of the cylinder using
       the inherited side as the radius.
7) Create Main Class:
    a. Make a separate class called Main for testing.
8) Test Square and Cylinder:
    a. Inside the Main class:
        i) Create objects of the Square and Cylinder classes.
        ii) Use these objects to call the getVolume methods, which will calculate and print the volumes of
            the square and cylinder.
```

Source Code:--

```
import java.lang.Math;
class Square {
   double side;
   public Square(double side) {
      this.side = side;
   }
   public double getVolume() {
      double volume = Math.pow(side, 2);
      System.out.println("Volume of Square: " + volume);
      return volume;
   }
}
class Cylinder extends Square {
   double height;
   public Cylinder(double side, double height) {
      super(side);
      this.height = height;
   }
   @Override
   public double getVolume() {
      double radius = side;
      double volume = Math.PI * Math.pow(radius, 2) * height;
      System.out.println("Volume of Cylinder: " + volume);
      return volume;
   }
}
```

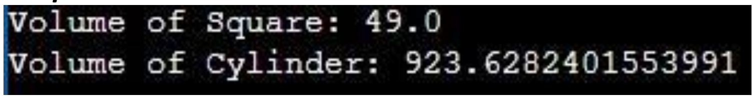
```
public class Main {
  public static void main(String[] args) {
    double sideLength = 4.0;
    double cylinderHeight = 2.0;
    Square squareObj = new Square(sideLength);
    squareObj.getVolume();
    Cylinder cylinderObj = new Cylinder(sideLength, cylinderHeight);
    cylinderObj.getVolume();
  }
}
```

Output 1:-



```
Volume of Square: 16.0
Volume of Cylinder: 100.53096491487338
```

Output 2:-



```
Volume of Square: 49.0
Volume of Cylinder: 923.6282401553991
```

Q7}Consider you are designing vehicles engine with 'speed:int, gear:int'. you can define your engine functionalities 'speedUp(value)' and 'changeGear(value) in an interface. The class which is implementing the interface should implement all the methods in the interface.

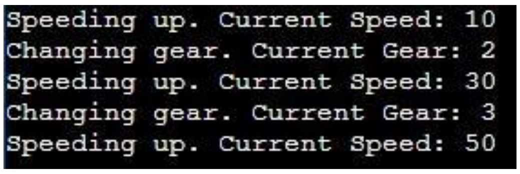
Algorithm :-

```
1) Initialise speed to 0 and gear to 1 (default gear).
2) Increase the speed attribute by the specified value.
        ○ Print a message indicating the current speed.
3) Set the gear attribute to the specified value.
        ○ Print a message indicating the gear change.
4) Main Method:
        ○ Invoke the speedUp method with a value to increase the speed.
        ○ Invoke the changeGear method with a value to change the gear.
        ○ Repeat these steps as needed to simulate engine operations.
```

Source Code :-

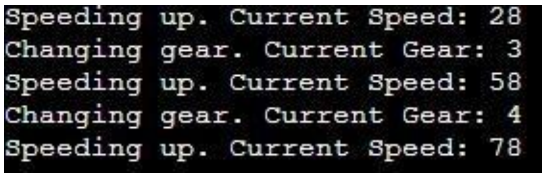
```
interface Engine {
 void speedUp(int value);
 void changeGear(int value);
}
class VehicleEngine implements Engine {
 private int speed;
 private int gear;
public VehicleEngine() {
 this.speed = 0;
 this.gear = 1; // Default gear is 1
 }
@Override
 public void speedUp(int value) {
 this.speed += value;
 System.out.println("Speeding up. Current Speed: " + this.speed);
 }
@Override
 public void changeGear(int value) {
 this.gear = value;
 System.out.println("Changing gear. Current Gear: " + this.gear);
 }
}
public class myproject {
 public static void main(String[] args) {
 VehicleEngine myCarEngine = new VehicleEngine();
 myCarEngine.speedUp(10);
 myCarEngine.changeGear(2);
 myCarEngine.speedUp(20);
 myCarEngine.changeGear(3);
 myCarEngine.speedUp(20);
 }
}
```

Output 1:-



```
Speeding up. Current Speed: 10
Changing gear. Current Gear: 2
Speeding up. Current Speed: 30
Changing gear. Current Gear: 3
Speeding up. Current Speed: 50
```

Output 2:-



```
Speeding up. Current Speed: 28
Changing gear. Current Gear: 3
Speeding up. Current Speed: 58
Changing gear. Current Gear: 4
Speeding up. Current Speed: 78
```

Q8}Write a program in Java that handles both 'ArrayIndexOutOfBoundsException' and 'ArithmeticException'.

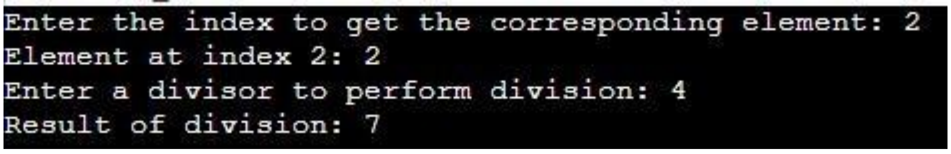
Algorithm :-

1) Initialize Scanner:
   - Create a Scanner object (scanner) for user input.
2) Try Block:
   - Enclose the code that might throw exceptions within a try block.
3) Handle ArrayIndexOutOfBoundsException:
   - Declare an array (numbers) with some elements.
   - Prompt the user to enter an index.
   - Call the getElementAt method with the array and index.
   - Print the result.
4) Handle ArithmeticException:
   - Prompt the user to enter a divisor.
   - Call the performDivision method with a constant dividend and the user-entered divisor.
   - Print the result.
5) Catch Block:
   - Catch ArrayIndexOutOfBoundsException and print an error message.
   - Catch ArithmeticException and print an error message.
   - Catch a general Exception and print an error message for unexpected exceptions.
6) Finally Block:
   - Close the Scanner (scanner) in the finally block to ensure it is always closed.
7) getElementAt Method:
   - Parameters: Takes an array (array) and an index (index).
   - Return: Returns the element at the specified index in the array.
8) performDivision Method:
   - Parameters: Takes a dividend (dividend) and a divisor (divisor).
   - Return: Returns the result of the division.

Source Code :-

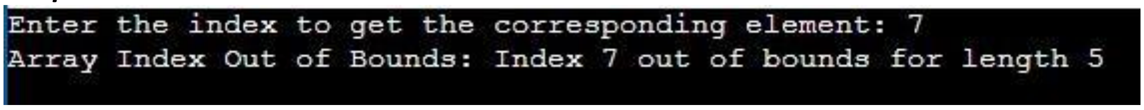
```
import java.util.Scanner;
public class ExceptionHandlingExample {
 public static void main(String[] args) {
 Scanner scanner = new Scanner(System.in);
 try {
 // Handle ArrayIndexOutOfBoundsException
 int[] numbers = {4, 3, 2, 1, 0};
 System.out.print("Enter the index to get the corresponding element: ");
 int index = scanner.nextInt();
 int result = getElementAt(numbers, index);
 System.out.println("Element at index " + index + ": " + result);
 // Handle ArithmeticException
 System.out.print("Enter a divisor to perform division: ");
 int divisor = scanner.nextInt();
 int quotient = performDivision(30, divisor);
 System.out.println("Result of division: " + quotient);
 } catch (ArrayIndexOutOfBoundsException e) {
 System.out.println("Array Index Out of Bounds: " + e.getMessage());
 } catch (ArithmeticException e) {
 System.out.println("Arithmetic Exception: " + e.getMessage());
 } catch (Exception e) {
 System.out.println("An unexpected exception occurred: " + e.getMessage());
 } finally {
 // Close the scanner in the finally block to ensure it always gets closed
 scanner.close();
 }
 }
 // Method to get an element at a specified index from an array
 private static int getElementAt(int[] array, int index) {
 return array[index];
 }
 // Method to perform division
 private static int performDivision(int dividend, int divisor) {
 return dividend / divisor;
 }
}
```

Output 1 :-



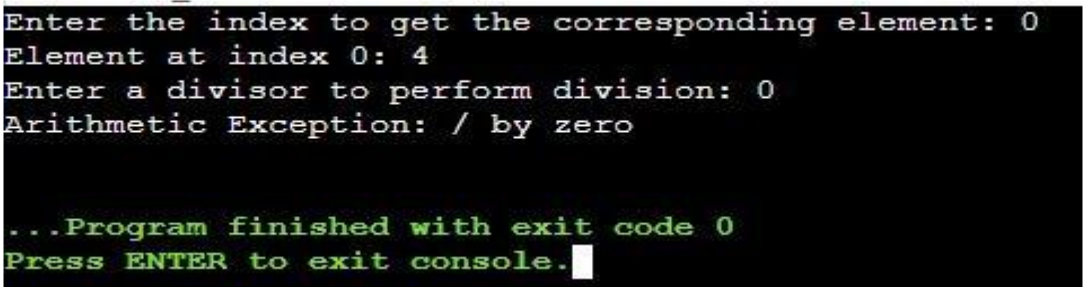
```
Enter the index to get the corresponding element: 2
Element at index 2: 2
Enter a divisor to perform division: 4
Result of division: 7
```

Output 2:-



```
Enter the index to get the corresponding element: 7
Array Index Out of Bounds: Index 7 out of bounds for length 5
```

Output 3:-



```
Enter the index to get the corresponding element: 0
Element at index 0: 4
Enter a divisor to perform division: 0
Arithmetic Exception: / by zero


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

Q9} Write a program in java that inputs students data(Regno, Sname, City, ContactNo) from user and insert into table 'Student_info' using JDBC connectivity. Also view all records in tabular format.

Algorithm :-

Step 1: Initialization
Set up the required imports for JDBC, Scanner, and other necessary classes.
Declare constants for the database connection details (url, username, password).

Step 2: Main Program Loop
Initialize a Scanner for user input.
Enter a while loop to create a menu-driven program.
Display menu options for the user.
Prompt the user to enter a choice (1-3).
Use a switch statement to perform actions based on the user's choice.

Step 3: Add New Information (addStudent)
Create a method addStudent that takes a Scanner as a parameter.
Prompt the user to enter student details (Registration Number, Student Name, City, Contact Number).
Use the entered details to call insertStudentInfo method.

Step 4: Insert Student Information (insertStudentInfo)
Create a method insertStudentInfo that takes parameters for Registration Number, Student Name, City, and Contact Number.
Establish a database connection using DriverManager.
Prepare an SQL query to insert data into the "Student_info" table using a PreparedStatement.
Set parameters in the prepared statement.
Execute the update and display success/failure messages.

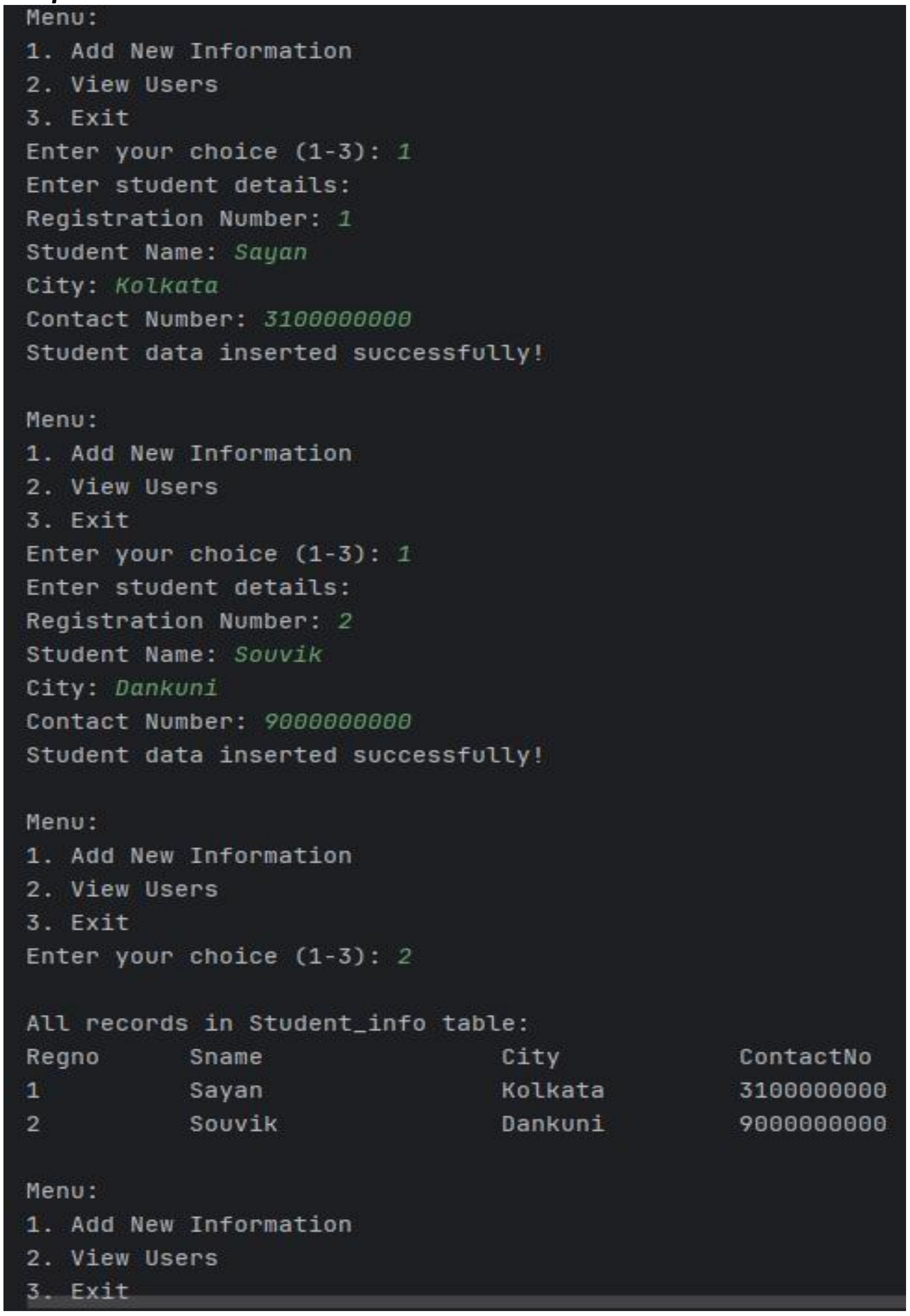
Step 5: View All Records (viewAllRecords)
Create a method viewAllRecords.
Establish a database connection using DriverManager.
Prepare an SQL query to select all records from the "Student_info" table using a Statement.
Execute the query, iterate through the result set, and display the records.

Step 6: Exception Handling
Implement exception handling for SQLException in both insertStudentInfo and viewAllRecords.
Print meaningful error messages to the console.

Step 7: Closing Resources and Exiting
Close the Scanner to prevent resource leaks.
Provide a clean exit from the program when the user chooses option 3.

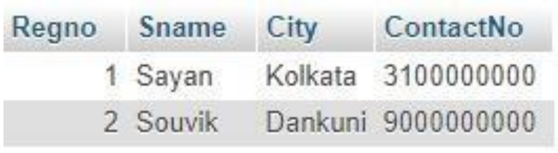
Source Code :-

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class jdbcDemo {
    private static final String url = "jdbc:mysql://localhost:3306/jdbc";
    private static final String username = "root";
    private static final String password = "";
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        while (true) {
            // Display menu
            System.out.println("\nMenu:");
            System.out.println("1. Add New Information");
            System.out.println("2. View Users");
            System.out.println("3. Exit");
            System.out.print("Enter your choice (1-3): ");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    addStudent(scanner);
                    break;
                case 2:
                    viewAllRecords();
                    break;
                case 3:
                    System.out.println("Exiting the program.");
                    System.exit(0);
                default:
                    System.out.println("Invalid choice. Please enter a valid option.");
            }
        }
    }
    private static void addStudent(Scanner scanner) {
        System.out.println("Enter student details:");
        System.out.print("Registration Number: ");
        int Regno = scanner.nextInt();
        scanner.nextLine(); // Consume newline
        System.out.print("Student Name: ");
        String Sname = scanner.nextLine();
        System.out.print("City: ");
        String City = scanner.nextLine();
        System.out.print("Contact Number: ");
        String ContactNo = scanner.nextLine();
        insertStudentInfo(Regno, Sname, City, ContactNo);
    }
    private static void insertStudentInfo(int Regno, String Sname, String City, String ContactNo) {
        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            String insertQuery = "INSERT INTO Student_info (Regno, Sname, City, ContactNo) VALUES (?, ?, ?, ?)";
            try (PreparedStatement preparedStatement = connection.prepareStatement(insertQuery)) {
                preparedStatement.setInt(1, Regno);
                preparedStatement.setString(2, Sname);
                preparedStatement.setString(3, City);
                preparedStatement.setString(4, ContactNo);
                int rowsAffected = preparedStatement.executeUpdate();
                if (rowsAffected > 0) {
                    System.out.println("Student data inserted successfully!");
                } else {
                    System.out.println("Failed to insert student data.");
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    private static void viewAllRecords() {
        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            String selectQuery = "SELECT * FROM Student_info";
            try (Statement statement = connection.createStatement();
                ResultSet resultSet = statement.executeQuery(selectQuery)) {
                System.out.println("\nAll records in Student_info table:");
                System.out.printf("%-10s %-20s %-15s %-15s\n", "Regno", "Sname", "City", "ContactNo");
                while (resultSet.next()) {
                    int Regno = resultSet.getInt("Regno");
                    String Sname = resultSet.getString("Sname");
                    String City = resultSet.getString("City");
                    String ContactNo = resultSet.getString("ContactNo");
                    System.out.printf("%-10d %-20s %-15s %-15s\n", Regno, Sname, City, ContactNo);
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
  }
```
Output :



```
Menu:
1. Add New Information
2. View Users
3. Exit
Enter your choice (1-3): 1
Enter student details:
Registration Number: 1
Student Name: Sayan
City: Kolkata
Contact Number: 3100000000
Student data inserted successfully!

Menu:
1. Add New Information
2. View Users
3. Exit
Enter your choice (1-3): 1
Enter student details:
Registration Number: 2
Student Name: Souvik
City: Dankuni
Contact Number: 9000000000
Student data inserted successfully!

Menu:
1. Add New Information
2. View Users
3. Exit
Enter your choice (1-3): 2

All records in Student_info table:
Regno      Sname                City           ContactNo
1          Sayan                Kolkata        3100000000
2          Souvik               Dankuni        9000000000

Menu:
1. Add New Information
2. View Users
3. Exit
```

Output of Database :-



Regno	Sname	City	ContactNo
1	Sayan	Kolkata	3100000000
2	Souvik	Dankuni	9000000000

Q10} Write a Java program to remove a specific element from an array.

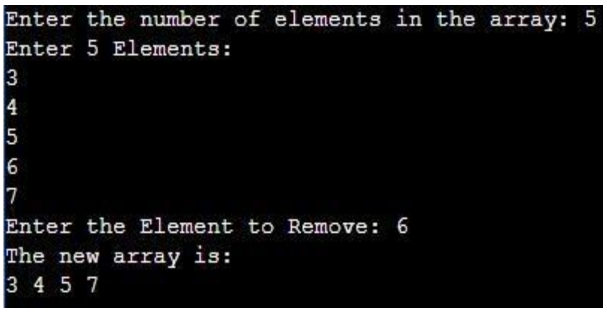
Algorithm:-

1. Initialize variables:
   Declare variables for loop control (i and j), array size (n), and the element to remove (element).
2. Input array size:
   Prompt the user to enter the number of elements in the array (n).
3. Check array size:
   Ensure that the array size (n) is greater than 0. If not, display an error message and end the program.
4. Input array elements:
   Use a loop to input n elements into the array (arr).
5. Input element to remove:
   Prompt the user to enter the element to be removed (element).
6. Remove the element:
   Use a loop to find the index of the specified element in the array and remove it by shifting elements to the left.
7. Display the modified array:
   Print the modified array after the specified element is removed.

Source Code :-

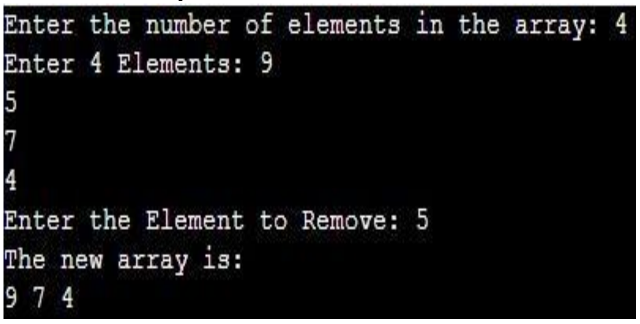
```
import java.util.Scanner;
public class arrRem {
    public static void main(String[] args) {
        int i, j, n, element;
        Scanner scan = new Scanner(System.in);
        // Input the number of elements in the array
        System.out.print("Enter the number of elements in the array: ");
        n = scan.nextInt();
        int[] arr = new int[n];
        // Input array elements
        System.out.print("Enter " + n + " Elements: ");
        for (i = 0; i < n; i++)
            arr[i] = scan.nextInt();
        System.out.print("Enter the Element to Remove: ");
        element = scan.nextInt();
      for(i=0; i<n; i++)
      {
         if(element==arr[i])
         {
            for(j=i; j<(n-1); j++)
               arr[j] = arr[j+1];
            break;
         }
      }
     System.out.println("The new array is: ");
      for(i=0; i<(n-1); i++)
         System.out.print(arr[i]+ " ");
   }
}
```

Output 1:-



```
Enter the number of elements in the array: 5
Enter 5 Elements:
3
4
5
6
7
Enter the Element to Remove: 6
The new array is:
3 4 5 7
```

Output 2 :-



```
Enter the number of elements in the array: 4
Enter 4 Elements: 9
5
7
4
Enter the Element to Remove: 5
The new array is:
9 7 4
```

Q11} Write a Java program to insert an element (specific position) into an array.

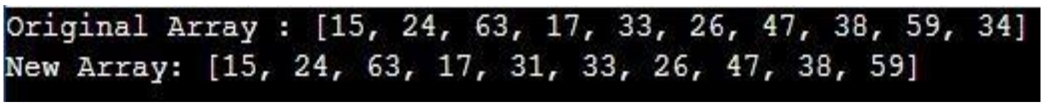
Algorithm :-

1. Initialize Array:
   An array my_array is initialized with given values.
2. Specify Insertion Position and Value:
   Index_position is set to 1 (for the 2nd position, considering index starts at 0).newValue is set to 95, the value to be inserted.
3. Display Original Array:
   The original array is displayed using Arrays.toString().
4. Shift Elements to Make Space:
   A loop starts from the end of the array and shifts each element to the right to make space for the new element.
5. Insert New Element:
   The new value (newValue) is inserted at the specified position (Index_position).
6. Display Modified Array:
   The modified array, after the insertion, is displayed using Arrays.toString().

Source Code:-

```
import java.util.Arrays;
public class arrInsert {
public static void main(String[] args) {
int[] my_array = {15, 24, 63, 17, 33, 26, 47, 38, 59, 34};
// Insert an element in 3rd position of the array (index->0, value->50)
int Index_position = 4;
int newValue = 31;
System.out.println("Original Array : "+Arrays.toString(my_array));
for(int i=my_array.length-1; i > Index_position; i--){
my_array[i] = my_array[i-1];
}
my_array[Index_position] = newValue;
System.out.println("New Array: "+Arrays.toString(my_array));
}
}
```

Algorithm :-



```
Original Array : [15, 24, 63, 17, 33, 26, 47, 38, 59, 34]
New Array: [15, 24, 63, 17, 31, 33, 26, 47, 38, 59]
```

Q12} Write a Java program to find all pairs of elements in an array whose sum is equal to a specified number.

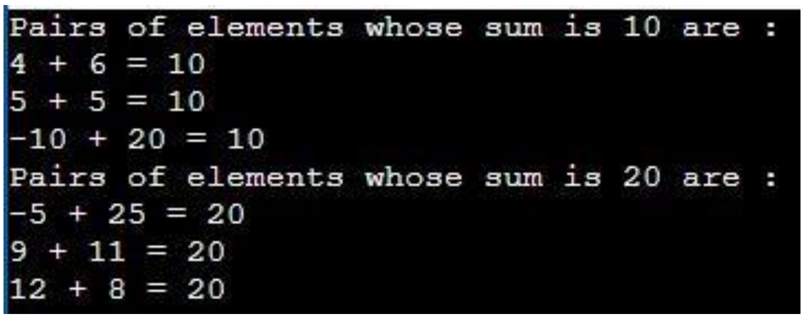
Algorithm :-

1.Define a class:
  Start by defining a class that encapsulates the functionality.
2.Create a static method named findThePairs:
  Implement a method named findThePairs that takes an array of integers (inputArray) and an integer (inputNumber) as parameters.
3. Use two nested loops to iterate through each pair of elements in the array:
  Employ two nested loops (using indices i and j) to consider each possible pair of elements in the inputArray.
4. Check if the sum of the current pair equals the given inputNumber:
  Inside the nested loops, use an if statement to check whether the sum of the current pair (inputArray[i] + inputArray[j]) is equal to the provided inputNumber.
5. If the sum condition is true, print the pair and their sum:
  If the condition in step 5 is satisfied, print the elements of the current pair and their sum.
6.Define the main method:
  Implement the main method to serve as the entry point for the program.
7.Call the findThePairs method with example arrays and target sums:
  Utilize the findThePairs method to find and print pairs of elements in two example arrays with specified target sums.
8.Close the main method and the class definition:
  Properly close the main method and the class definition to complete the program.

Source Code :-

```
class s{
   static void findThePairs(int inputArray[], int inputNumber) {
      System.out.println("Pairs of elements whose sum is "+inputNumber+" are : ");
      for (int i = 0; i < inputArray.length; i++) {
         for (int j = i+1; j < inputArray.length; j++){
            if(inputArray[i]+inputArray[j] == inputNumber) {
               System.out.println(inputArray[i]+" + "+inputArray[j]+" = "+inputNumber);
            }
         }
      }
   }
   public static void main(String[] args)
   {
      findThePairs(new int[] {4, 6, 5, -10, 8, 5, 20}, 10);

      findThePairs(new int[] {4, -5, 9, 11, 25, 13, 12, 8}, 20);
   }
}
```

Output :-



```
Pairs of elements whose sum is 10 are :
4 + 6 = 10
5 + 5 = 10
-10 + 20 = 10
Pairs of elements whose sum is 20 are :
-5 + 25 = 20
9 + 11 = 20
12 + 8 = 20
```

Q13} Write a Java program to remove the duplicate elements of a given array and return the new length of that array.

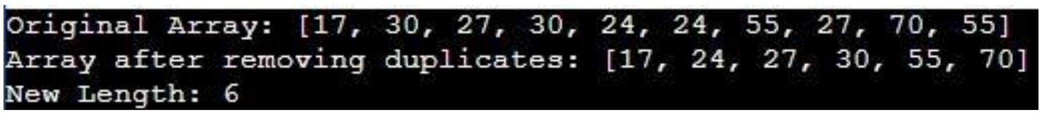
Algorithm :-

```
1. Create a method removeDuplicates that takes an integer array as input and returns the new length after removing duplicates.
2. Check if the length of the array is 0 or 1. If true, return the length of the array as there are no duplicates to remove.
3. Sort the array in ascending order using Arrays.sort.
4. Initialize a variable uniqueIndex to 1. This variable will keep track of the unique elements in the modified array.
5. Iterate through the array starting from index 1.
    If the current element is different from the previous element, update the element at uniqueIndex with the current element, and increment uniqueIndex.
6. After the loop, the elements up to the uniqueIndex represent the unique elements in the array.
7. Return the value of uniqueIndex as the new length of the array.
```

Source Code :-

```java
import java.util.Arrays;
public class rem {
   public static void main(String[] args) {
      int[] arr = {17, 30, 27, 30, 24, 24, 55, 27, 70, 55};
      System.out.println("Original Array: " + Arrays.toString(arr));
      int newLength = removeDuplicates(arr);
      System.out.println("Array after removing duplicates: " + Arrays.toString(Arrays.copyOf(arr,
newLength)));
      System.out.println("New Length: " + newLength);
   }
   private static int removeDuplicates(int[] arr) {
      if (arr.length == 0 || arr.length == 1) {
         return arr.length;
      }
      Arrays.sort(arr);
      int uniqueIndex = 1;
      for (int i = 1; i < arr.length; i++) {
         if (arr[i] != arr[i - 1]) {
            arr[uniqueIndex] = arr[i];
            uniqueIndex++;
         }
      }
      return uniqueIndex;
}
}
```

Output :-



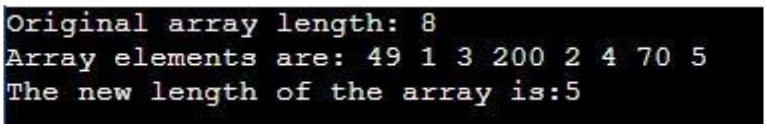
```
Original Array: [17, 30, 27, 30, 24, 24, 55, 27, 70, 55]
Array after removing duplicates: [17, 24, 27, 30, 55, 70]
New Length: 6
```

14} Write a Java program to find the length of the longest consecutive elements sequence from a given unsorted array of integers.
Sample array: [49, 1, 3, 200, 2, 4, 70, 5]
The longest consecutive elements sequence is [1, 2, 3, 4, 5], therefore the program will return its length 5.

Algorithm :-
1. Create a HashSet to store the unique elements of the array for efficient lookups.
2. Iterate through the array and add each element to the HashSet.
3. For each element in the array, check if the element minus one is not present in the HashSet. This condition indicates the potential start of a consecutive sequence.
4. If the above condition is true, initiate a loop to iterate forward and find the length of the consecutive sequence. Keep incrementing the current element until the next consecutive element is not present in the HashSet.
5. Track and update the maximum length encountered during the iteration.
6. Repeat steps 3-5 for all elements in the array.
7. Return the maximum length as the result.

Source Code :-
```
import java.util.HashSet;
public class cons {
public static void main(String[] args) {
int nums[] = {49, 1, 3, 200, 2, 4, 70, 5};
System.out.println("Original array length: "+nums.length);
System.out.print("Array elements are: ");
for (int i = 0; i < nums.length; i++)
{
System.out.print(nums[i]+" ");
}
System.out.println("\nThe new length of the array is:"+longest_sequence(nums));
}
public static int longest_sequence(int[] nums) {
final HashSet<Integer> h_set = new HashSet<Integer>();
for (int i : nums) h_set.add(i);
int longest_sequence_len = 0;
for (int i : nums) {
int length = 1;
for (int j = i - 1; h_set.contains(j); --j) {
h_set.remove(j);
++length;
}
for (int j = i + 1; h_set.contains(j); ++j) {
h_set.remove(j);
++length;
}
longest_sequence_len = Math.max(longest_sequence_len, length);
}
return longest_sequence_len;
}
}
```

Output :-


```
Original array length: 8
Array elements are: 49 1 3 200 2 4 70 5
The new length of the array is:5
```

15} Write a java program to compare two strings lexicographically.

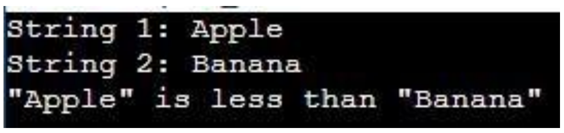
Algorithm :-

```
1.Import the necessary libraries, in this case, java.util.Scanner.
2.Create a Scanner object to read input from the user.
3.Prompt the user to input the first string.
4.Prompt the user to input the second string.
5.Use the compareTo method of the String class to compare the two strings lexicographically.
6.Store the  result in the variable result.
7.Check the value of result.
    (i)If result is less than 0, print that the first string is lexicographically smaller.
    (ii)If result is greater than 0, print that the first string is lexicographically larger.
    (iii)If result is equal to 0, print that both strings are lexicographically equal.
8.Close the Scanner to release resources.
```

Source Code :-

```java
public class lex {
    public static void main(String[] args) {
        // Change the values of str1 and str2
        String str1 = "Apple";
        String str2 = "Banana";

        System.out.println("String 1: " + str1);
        System.out.println("String 2: " + str2);

        // Compare the two strings.
        int result = str1.compareTo(str2);

        // Display the results of the comparison.
        if (result < 0) {
            System.out.println("\"" + str1 + "\"" + " is less than " + "\"" + str2 + "\"");
        } else if (result == 0) {
            System.out.println("\"" + str1 + "\"" + " is equal to " + "\"" + str2 + "\"");
        } else { // if (result > 0)
            System.out.println("\"" + str1 + "\"" + " is greater than " + "\"" + str2 + "\"");
        }
    }
}
```

Output :-



```
String 1: Apple
String 2: Banana
"Apple" is less than "Banana"
```

16} Write a Java program to find whether a region in the current string matches a region in another string.
Sample Output:
str1[0 - 7] == str2[28 - 35]? true
str1[9 - 15] == str2[9 - 15]? False

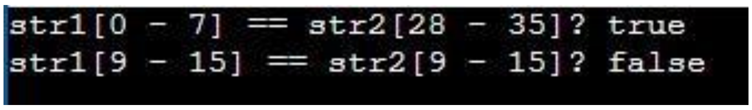
Algorithm :-

- Main Method:
  - Declare and initialize two strings, str1 and str2, with sample values.
  - Call the compareRegions method twice with different input parameters.
  - Print the results of the comparisons.
- compareRegions Method:
  - This method takes two strings (str1 and str2) and the start and end indices of the regions to be compared in each string.
  - It first checks if the lengths of the specified regions are equal. If not, the method returns false since regions of different lengths cannot be equal.
  - If the lengths are equal, the method iterates over the characters in the specified regions of both strings.
  - It compares each character in the corresponding positions of the two regions.
  - If any pair of characters are not equal, the method returns false since the regions are not identical.
  - If all characters are equal, the method returns true indicating that the specified regions in the two strings are the same**.**

Source Code :-

```
public class region
{
 public static void main (String[]args)
 {
  String str1 = "Apple Banana Mango Grapes";

  String str2 = "Grapes Mango Banana Apple";
  boolean match1 = str1.regionMatches (0, str2, 20, 5);
  boolean match2 = str1.regionMatches (9, str2, 13, 5);
// Display the results of the regionMatches method calls.
   System.out.println ("str1[0 - 7] == str2[28 - 35]? " + match1);
   System.out.println ("str1[9 - 15] == str2[9 - 15]? " + match2);

 }
}
```

Output :-



```
str1[0 - 7] == str2[28 - 35]? true
str1[9 - 15] == str2[9 - 15]? false
```

17} Write a Java program to print all permutations of a given string with repetition.
Sample Output:
The given string is: PQR
The permuted strings are:
PPP, PPQ, PPR, RRP, RRQ, RRR

Algorithm :-

1. Input: Read a string from the user.
2. Print the input string.
3. Initialize the permutation generation:

· Define a recursive function printPermutations.
· If the length of the current permutation is equal to the length of the input string, print the current permutation.
· Otherwise, for each character in the input string, append it to the current permutation and recursively call the printPermutations function.

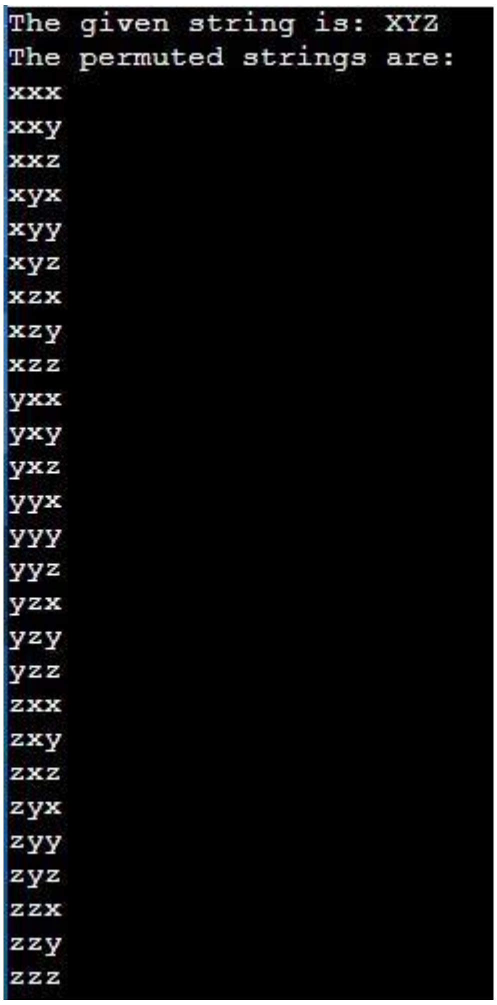
4. Output: Display the generated permutations.

Here's a step-by-step breakdown:
· The program starts by reading a string from the user.
· It then prints the input string to the console.
· The printPermutations method is called with the initial parameters: the input string and an empty string for the current permutation.
· Inside the printPermutations method:
· If the length of the current permutation is equal to the length of the input string, it prints the current permutation.
· Otherwise, for each character in the input string, it appends the character to the current permutation and makes a recursive call to printPermutations with the updated permutation.
· The process continues until all permutations are generated and printed.
· The output shows the input string and the list of generated permutations with repetition.

Source Code :-

```
import java.util.*;
public class per
{
 public static void main (String[]args)
 {
  permutationWithRepeation ("xyz");
 }
 private static void permutationWithRepeation (String str1)
 {
  System.out.println ("The given string is: XYZ");
  System.out.println ("The permuted strings are:");
  showPermutation (str1, "");
 }
 private static void showPermutation (String str1, String NewStringToPrint)
 {
  if (NewStringToPrint.length () == str1.length ())
    {
        System.out.println (NewStringToPrint);
        return;
    }
  for (int i = 0; i < str1.length (); i++)
    {
        showPermutation (str1, NewStringToPrint + str1.charAt (i));
    }
 }
}
```

Output :-


```
The given string is: XYZ
The permuted strings are:
xxx
xxy
xxz
xyx
xyy
xyz
xzx
xzy
xzz
yxx
yxy
yxz
yyx
yyy
yyz
yzx
yzy
yzz
zxx
zxy
zxz
zyx
zyy
zyz
zzx
zzy
zzz
```

Q18} Write a Java method to count all words in a string.
Test Data:
Input the string: The quick brown fox jumps over the lazy dog.
Expected Output:
Number of words in the string: 9

Algorithm :-

- Start:
  - Initialize a variable count to 0 to keep track of the number of words.
  - Initialize a boolean variable isWord to false, indicating whether the current character is part of a word.
- Input:
  - Accept the input string from the user.
- Iterate through each character in the string:
  - Start a loop that goes through each character of the input string.
- For each character at index i:
  - Retrieve the character at index i and store it in a variable ch.
- Check if the character is not a space:
- If ch is not equal to a space (' '):
- Check if isWord is false (indicating the start of a new word):
  - Increment the count by 1, indicating the start of a new word.

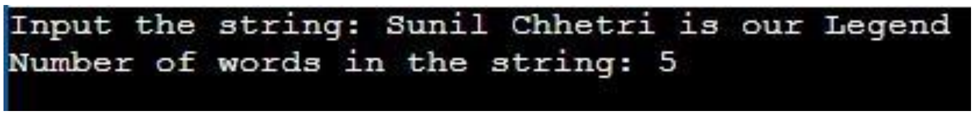
Set isWord to true, indicating that the current character is part of a word.

- Check if the character is a space:
- If ch is equal to a space (' '):
  - Set isWord to false, indicating the end of the current word.
- Continue the loop:
  - Continue the loop until all characters in the string are processed.
- Output:
  - Print the final value of count, which represents the total number of words in the input string.
- End.

Source Code :-

```
import java.util.Scanner;
public class count{
public static void main (String[]args) {
Scanner in = new Scanner (System.in);
System.out.print ("Input the string: ");
String str = in.nextLine ();
System.out.print ("Number of words in the string: " + count_Words (str) + "\n");
}
public static int count_Words (String str) {
int count = 0;
if (!(" ".equals (str.substring (0, 1))) || !("".equals (str.substring (str.length () - 1)))) {
for (int i = 0; i < str.length (); i++)  {
if (str.charAt (i) == ' ' {
count++;
}
}
count = count + 1;
}
return count;                  // returns 0 if string starts or ends with space " ".
  }
}
```

Output :-



```
Input the string: Sunil Chhetri is our Legend
Number of words in the string: 5
```

19} Write a program in java to create Box class with parameterized constructor with an object argument to initialize length, breadth and height also create a function volume which returns the volume of the box and print it in main method.

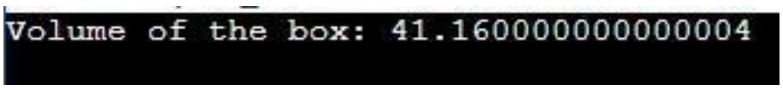
Algorithm :-

1. **Define the Box class:**
   Declare a class named Box.
2. **Declare private instance variables:**
   Declare private instance variables for length, breadth, and height within the Box class.
3. **Write the parameterized constructor:**
   Define a parameterized constructor that takes three parameters (length, breadth, and height).
   Inside the constructor, assign the passed values to the corresponding instance variables.
4. **Create a method to calculate volume:**
   Define a method named calculateVolume that calculates and returns the volume of the box using Length*breadth*height formula.
5. **Implement the main method:**
   Declare the main method.
6. **Inside the main method**:
   Create an object of the Box class using the new keyword and the parameterized constructor.
   Pass specific values for length, breadth, and height when creating the object.
7. **Calculate and print the volume:**
   Call the calculateVolume method on the created Box object to calculate the volume.
   Print the calculated volume to the console.
8. **End of the program:**
   End the program.

Source Code :-

```
public  class Box {
   private double length;
   private double breadth;
   private double height;
   // Parameterized constructor
   public Box(double length, double breadth, double height) {
      this.length = length;
      this.breadth = breadth;
      this.height = height;
   }
   // Method to calculate the volume of the box
   public double calculateVolume() {
      return length * breadth * height;
   }
   public static void main(String[] args) {
      // Creating an object of Box class using the parameterized constructor
      Box myBox = new Box(3.5, 4.2, 2.8);
      // Calculating and printing the volume of the box
      double volume = myBox.calculateVolume();
      System.out.println("Volume of the box: " + volume);
   }
}
```

Output :-



```
Volume of the box: 41.160000000000004
```

20} Write a program in java with class Employee and do the following operations on it
a) Create two constructor default and with Object as parameter to initialize class variables.
b) Create a function Calculate which calculates the pf and allowances on the salary of employee and return the all values as an object.

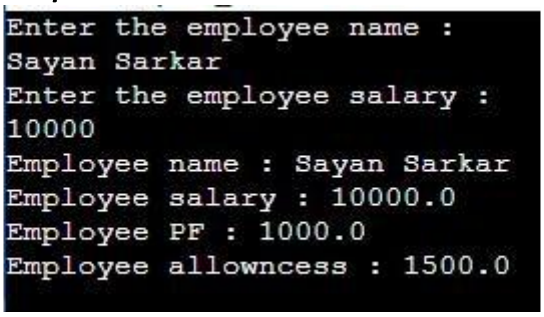
Algorithm :-

1. Create a class called Employee.
2. Define the following variables: name, salary, pf, allowences
3. Define a parameterized constructor that takes an object of the Employeeclass as an argument.
4. In the constructor, initialize the name, salary of the current employee with the values of the passed object.
5. Define a function called calculate that calculate pf and allowences of a employee.
6. Define a function that returns pf of salary of a employee.
7. Define a function that returns allowences of salary of a employee.

Source Code :-

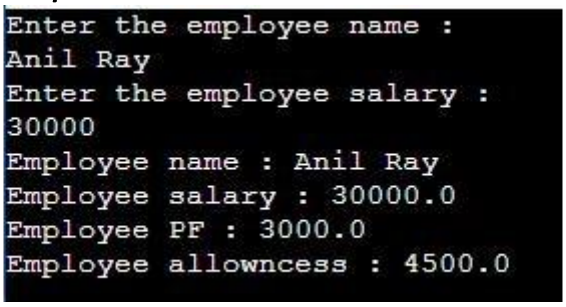
```
import java.util.Scanner;
public class salCal {
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter the employee name :");
String name= sc.nextLine();
System.out.println("Enter the employee salary : ");
double salary = sc.nextDouble();
// initialize
Employee employee1 = new Employee(name, salary);
//calling
employee1.calculate();
System.out.println("Employee name : "+employee1.getname());
System.out.println("Employee salary : "+employee1.getSalary());
System.out.println("Employee PF : " + employee1.getPF());
System.out.println("Employee allowncess : " + employee1.getAllowances());
}
}
class Employee {
// Class variables
private String name;
private double salary;
private double pf;
private double allowances;
// Default constructor
public Employee() {
this.name = "employee";
this.salary = 0.0;
}
// Constructor with parameters
public Employee(String name, double salary) {
this.name = name;
this.salary = salary;
}
// Method to calculate PF and allowances
public void calculate() {
this.pf = 0.1 * this.salary; // Assuming PF is 10% of the salary
this.allowances = 0.15 * this.salary; // Assuming allowances is 15% of the salary
}
// get name
public String getname() {
return this.name;
}
// get PF
public double getPF() {
return this.pf;
}
// get allowances
public double getAllowances() {
return this.allowances;
}
// get salary
public double getSalary() {
return this.salary;
}
}
```

Output 1:-



```
Enter the employee name :
Sayan Sarkar
Enter the employee salary :
10000
Employee name : Sayan Sarkar
Employee salary : 10000.0
Employee PF : 1000.0
Employee allowncess : 1500.0
```

Output 2:-



```
Enter the employee name :
Anil Ray
Enter the employee salary :
30000
Employee name : Anil Ray
Employee salary : 30000.0
Employee PF : 3000.0
Employee allowncess : 4500.0
```

21} Write a program to create your own exception as NegativeSizeException whenever negative values are put in an array.

Algorithm :-

- **Import required packages**:
  - Import the necessary packages for input/output operations (java.io.BufferedReader, java.io.IOException, java.io.InputStreamReader).
- **Define custom exception class:**
  - Create a custom exception class MyException that extends the Exception class.
  - Provide a constructor that takes a string as a parameter and calls the superclass constructor with the given message.
- **Define the main method:**
  - Declare the main method with the throws IOException clause.
- **Initialize BufferedReader for input:**
  - Create a BufferedReader object (br) to read input from the user using InputStreamReader.
- **Prompt user for input:**
  - Display the message "Input number ::" to prompt the user to enter a number.
- **Try block for exception handling:**
  - Inside the try block:
  - Read an integer from the user and parse it using Integer.parseInt.
  - If the entered number is less than 0, throw a MyException with the message "Number is negative."
  - If the entered number is greater than or equal to 0, throw a MyException with the message "Number is positive."
- **Catch block for handling the custom exception:**
  - In the catch block:

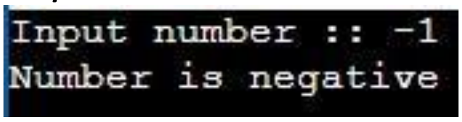
    Catch the MyException and print the exception message using getMessage().
- **End of the program:**

  End the program.

Source Code :-

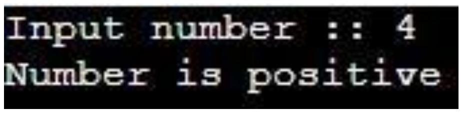
```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
class MyException extends Exception {
   public MyException(String str) {
      super(str);
   }
}
public class negEx {
   public static void main(String[] args) throws IOException {
      BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
      System.out.print("Input number :: ");
      try {
         int num = Integer.parseInt(br.readLine());
         if (num < 0)
            throw new MyException("Number is negative");
         else
            throw new MyException("Number is positive");
      } catch (MyException m) {
         System.out.println(m.getMessage()); // Use getMessage() to print the exception message.
      }
   }
}
```

Output 1 :-



```
Input number :: -1
Number is negative
```

Output 2:-



```
Input number :: 4
Number is positive
```

22} Create a class Student with following operations
a) create parameterized constructor to initialize the objects.
b) create a function isEqual() to check whether the two objects are equal or not which returns the Boolean value and gets two objects.
c) print the result in main method if objects are equals or not (take variables as your assumption)

Algorithm :-

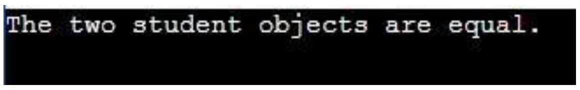
1. Create a class called Student with the following attributes:
   o name
   o age
   o grade

2. Create a parameterized constructor that takes the name, age, and grade of the student as parameters and initializes the attributes accordingly.
3. Create a function called isEqual() that takes two Student objects as parameters and returns True if the two objects have the same name, age, and grade, and False otherwise.
4. To use the Student class, we can create instances of the class and call the isEqual() function to compare two objects.

Source Code :-

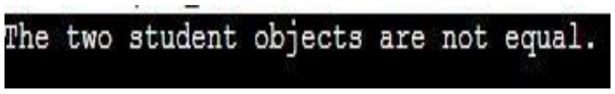
```
public class objeq{
  public static void main (String[]args){
// Creating two Student objects
    Student student1 = new Student ("Sayan Sarkar", 2100082, 21);
    Student student2 = new Student ("Sayan Sarkar", 2100082, 21);
// Checking equality using isEqual method
    boolean areEqual = student1.isEqual (student2);
// Printing the result
    if (areEqual){
            System.out.println ("The two student objects are equal.");
      }
    else{
            System.out.println ("The two student objects are not equal.");
      }
  }
}

class  Student{
  private String name;
  private int rollNumber;
  private int age;
// Parameterized constructor
  public Student (String name, int rollNumber, int age) {
    this.name = name;
    this.rollNumber = rollNumber;
    this.age = age;
  }
// isEqual method to check equality
  public boolean isEqual (Student otherStudent){
// Compare attributes of two objects to check equality
    return this.name.equals (otherStudent.name)
      &&
      this.rollNumber == otherStudent.rollNumber
      && this.age == otherStudent.age;
  }
}
```

Output 1 :-



```
The two student objects are equal.
```

Output 2:-



```
The two student objects are not equal.
```

23} Create an abstract class employee, having its properties and abstract function for calculating net salary and displaying the information. Derive manager and clerk class from this abstract class and implement the abstract method net salary and override the display method.

Algorithm :-

Abstract Employee Class:

- Define an abstract class named Employee.
- Declare instance variables name and basicSalary.
- Provide a constructor to initialize these variables.
- Declare two abstract methods: calculateNetSalary() and displayInformation().

Manager Class (extends Employee):

- Define a class named Manager that extends the Employee class.
- Add an additional variable allowance specific to the manager.
- Provide a constructor to initialize the variables, including calling the constructor of the superclass (Employee).
- Override the abstract methods calculateNetSalary() and displayInformation() with manager-specific implementations.

Clerk Class (extends Employee):

- Define a class named Clerk that extends the Employee class.
- Add an additional variable overtimeHours specific to the clerk.
- Provide a constructor to initialize the variables, including calling the constructor of the superclass (Employee).
- Override the abstract methods calculateNetSalary() and displayInformation() with clerk-specific implementations.

Main Class:

- Define a Main class to serve as the entry point of the program.

In the main method:

- Create an instance of the Manager class with specific values.
- Display the information of the manager using the displayInformation() method.
- Create an instance of the Clerk class with specific values.
- Display the information of the clerk using the displayInformation() method.

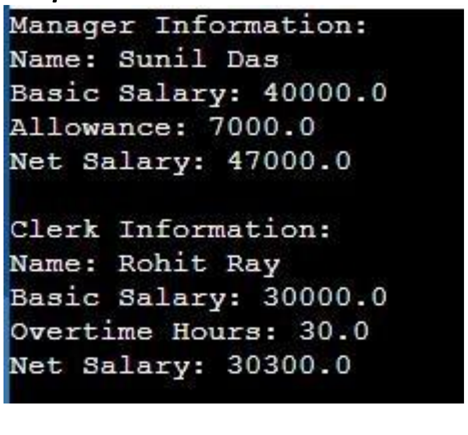
Output:

When the program is executed, it will display information about a manager and a clerk, including their names, basic salaries, and net salaries.

Source Code :-

```
abstract class Employee {
  protected String name;
  protected double basicSalary;
  public Employee(String name, double basicSalary) {
    this.name = name;
    this.basicSalary = basicSalary;
  }
  // Abstract method to calculate net salary
  public abstract double calculateNetSalary();
  // Abstract method to display information
  public abstract void displayInformation();
}
class Manager extends Employee {
  private double allowance;
  public Manager(String name, double basicSalary, double allowance) {
    super(name, basicSalary);
    this.allowance = allowance;
  }
  // Override calculateNetSalary method for Manager
  @Override
  public double calculateNetSalary() {
        return basicSalary + allowance;
    }
    // Override displayInformation method for Manager
    @Override
    public void displayInformation() {
        System.out.println("Manager Information:");
        System.out.println("Name: " + name);
        System.out.println("Basic Salary: " + basicSalary);
        System.out.println("Allowance: " + allowance);
        System.out.println("Net Salary: " + calculateNetSalary());
    }
}
class Clerk extends Employee {
    private double overtimeHours;

    public Clerk(String name, double basicSalary, double overtimeHours) {
        super(name, basicSalary);
        this.overtimeHours = overtimeHours;
    }
    // Override calculateNetSalary method for Clerk
    @Override
    public double calculateNetSalary() {
        // For simplicity, assuming overtime payment is hourly
        double overtimePaymentRate = 10.0; // Example rate per hour
        double overtimePayment = overtimeHours * overtimePaymentRate;
        return basicSalary + overtimePayment;
    }
    // Override displayInformation method for Clerk
    @Override
    public void displayInformation() {
        System.out.println("Clerk Information:");
        System.out.println("Name: " + name);
        System.out.println("Basic Salary: " + basicSalary);
        System.out.println("Overtime Hours: " + overtimeHours);
        System.out.println("Net Salary: " + calculateNetSalary());
    }
}
public class sal {
    public static void main(String[] args) {
        Manager manager = new Manager("Sunil Das", 40000, 7000);
        manager.displayInformation();
        System.out.println(); // Just for spacing
        Clerk clerk = new Clerk("Rohit Ray", 30000, 30);
        clerk.displayInformation();
}
}
```

Output :-



```
Manager Information:
Name: Sunil Das
Basic Salary: 40000.0
Allowance: 7000.0
Net Salary: 47000.0

Clerk Information:
Name: Rohit Ray
Basic Salary: 30000.0
Overtime Hours: 30.0
Net Salary: 30300.0
```