Ques 1:-

Create a simple Java program that models a Book. The Book class should have the following properties:

- `title` (String)
- `author` (String)
- `price` (double)

The Book class should have:

1. A constructor to initialise all the properties.
2. Getter and setter methods for each property.
3. A method to display the details of the book.

Write a main method to create an instance of the Book class, set its properties, and display its details.

Solution:

```
public class OOps {

    public static void main(String[] args) {

        Book b1 = new Book("3 idiots","chetan bhagat",400);

        b1.displayBooks();

    }

}

class Book{

    private String title;

    private String author;

    private double price;


    public Book(String title,String author,double price){

        this.title = title;
```

```
    this.author = author;
    this.price  =price;



}
public String getTitle(){
    return title;
}
public void setName( String title){
    this.title = title;
}
public String getAuthor(){
    return author;
}
public void setAuthor(String author){
    this.author = author;
}
public double  getPrice(){
    return price;
}
public void setPrice(double price){
    this.price = price;
}
public void displayBooks(){
    System.out.println("Title " + title);
    System.out.println("Author " + author);
```

```
        System.out.println("Price " + price);
    }
}
```

Ques 2 -

Create a simple Java program that models a Student. The Student class should have the following properties:

- name (String)
- rollNumber (int)
- marks (double)

The Student class should have:

1. A constructor to initialize all the properties.
2. Getter and setter methods for each property.
3. A method to display the details of the student.
4. A method to calculate the grade based on the marks:
    - 90 and above: A
    - 80 to 89: B
    - 70 to 79: C
    - 60 to 69: D
    - Below 60: F

Write a main method to create an instance of the Student class, set its properties, and display its details along with the grade.

Solution:

```
public class Main {
  public static void main(String[] args) {
     Student s1 = new Student("John",45,55);
     s1.displayDetails();
  }
}
class Student{
  private String name;
  private int roll_no;
  private double marks;

  public Student(String name,int roll_no,double marks){
     this.name = name;
     this.roll_no = roll_no;
     this.marks = marks;
  }
  public String getName(){
```

```
        return name;
    }
    public void setName(String name){
        this.name = name;
    }
    public void displayDetails(){
        System.out.println(name);
        System.out.println(roll_no);
        System.out.println(marks);
        System.out.println(calculateGrade());
    }
    public String calculateGrade(){
        if(marks>= 90){
            return "A";
        } else if (marks>=80) {
            return "B";
        }else if (marks>=70) {
            return "C";
        }else if (marks>=60) {
            return "D";
        }else {
            return "F";
        }
    }
}
```

Qus3:

Create a Java program that models a Person and a Student. The Student class should inherit from the Person class.
The Person class should have the following properties:

- name (String)
- age (int)

The Person class should have:

1. A constructor to initialise the properties.
2. Getter and setter methods for each property.
3. A method to display the details of the person.

The Student class should have the following additional properties:

- rollNumber (int)
- marks (double)

The Student class should have:

1. A constructor to initialise all properties, including those from the Person class.
2. Getter and setter methods for each property.
3. A method to display the details of the student.
4. A method to calculate the grade based on the marks:
    - 90 and above: A

    - 80 to 89: B
    - 70 to 79: C
    - 60 to 69: D
    - Below 60: F

Write a main method to create an instance of the Student class, set its properties, and display its details along with the grade.

Solution

```
class Person {
  // Properties
  private String name;
  private int age;

  // Constructor
  public Person(String name, int age) {
     this.name = name;
     this.age = age;
  }

  // Getter for name
  public String getName() {
     return name;
  }

  // Setter for name
  public void setName(String name) {
     this.name = name;
  }

  // Getter for age
  public int getAge() {
     return age;
  }

  // Setter for age
  public void setAge(int age) {
     this.age = age;
  }

  // Method to display person details
  public void displayDetails() {
     System.out.println("Name: " + name);
     System.out.println("Age: " + age);
  }
}
```

```
class Student extends Person {
    // Additional properties
    private int rollNumber;
    private double marks;

    // Constructor
    public Student(String name, int age, int rollNumber, double marks) {
        super(name, age); // Call the constructor of the Person class
        this.rollNumber = rollNumber;
        this.marks = marks;
    }

    // Getter for rollNumber
    public int getRollNumber() {
        return rollNumber;
    }

    // Setter for rollNumber
    public void setRollNumber(int rollNumber) {
        this.rollNumber = rollNumber;
    }

    // Getter for marks
    public double getMarks() {
        return marks;
    }

    // Setter for marks
    public void setMarks(double marks) {
        this.marks = marks;
    }

    // Method to display student details
    @Override
    public void displayDetails() {
        super.displayDetails(); // Call the displayDetails method of the Person class
        System.out.println("Roll Number: " + rollNumber);
        System.out.println("Marks: " + marks);
        System.out.println("Grade: " + calculateGrade());
    }

    // Method to calculate grade
    public String calculateGrade() {
        if (marks >= 90) {
            return "A";
        } else if (marks >= 80) {
            return "B";
```

```
        } else if (marks >= 70) {
            return "C";
        } else if (marks >= 60) {
            return "D";
        } else {
            return "F";
        }
    }
}

public class Main {
    public static void main(String[] args) {
        // Create an instance of Student
        Student student = new Student("Mark", 40, 101, 85.5);

        // Display student details
        student.displayDetails();
    }
}
```

Ques4 :-

E-Commerce Application

You are tasked with designing a simple e-commerce application. The application should have the following classes and functionality:

1. Product: A base class representing a generic product.
    - Fields:
        - productId (int)
        - productName (String)
        - price (double)
    - Methods:
        - Constructors to initialise the fields.
        - Getters and setters for each field.
        - displayProductDetails() method to display the product details.
2. Electronics: A subclass of Product representing electronic products.
    - Additional Fields:
        - brand (String)
        - warranty (int) - warranty period in months
    - Methods:
        - Constructor to initialise all fields, including those in the superclass.
        - Getters and setters for the additional fields.
        - Override displayProductDetails() to include brand and warranty information.
3. Clothing: A subclass of Product representing clothing items.
    - Additional Fields:
        - size (String)

        - material (String)
    - Methods:
        - Constructor to initialise all fields, including those in the superclass.
        - Getters and setters for the additional fields.
        - Override displayProductDetails() to include size and material information.
4. ShoppingCart: A class representing a shopping cart.
    - Fields:
        - cartItems (ArrayList of Product)
    - Methods:
        - addProduct(Product product): Adds a product to the cart.
        - removeProduct(int productId): Removes a product from the cart based on productId.
        - displayCartDetails(): Displays details of all products in the cart.
        - calculateTotal(): Calculates and returns the total price of all products in the cart.

Requirements:

1. Implement the Product, Electronics, Clothing, and ShoppingCart classes with the specified fields and methods.
2. Demonstrate the usage of these classes in a main method:
    - Create instances of Electronics and Clothing.
    - Add these products to the ShoppingCart.
    - Display the cart details.
    - Calculate and display the total price of products in the cart.
    - Remove a product from the cart and display the updated cart details.

Ques-5

Explanation

1. **Class Definition: The `BookCD` class encapsulates the properties (title, author/artist, price, type) and behaviors of a Book or CD.**
2. **Constructor: Initialises the `BookCD` object with the specified attributes.**
3. **Methods:**
    - **`displayDetails`: Prints the details of the Book or CD.**
    - **`applyDiscount`: Applies a discount to the price if the discount percentage is valid (between 0 and 100).**

    - **Getter and Setter methods for title, authorOrArtist, price, and type: Allows for retrieving and updating these attributes.**
4. **Main Method: Demonstrates creating `BookCD` objects for both a book and a CD, displaying their details, and applying a discount to their prices.**

Ques 6:-

Explanation

1. **Class Definition**: The `Box` class encapsulates the properties (length, width, height) and behaviours of a box.
2. **Constructor**: Initialises the box with the specified dimensions.
3. **Methods**:
    - `calculateVolume`: Calculates and returns the volume of the box using the formula `length * width * height`.
    - `calculateSurfaceArea`: Calculates and returns the surface area of the box using the formula `2 * (length * width + width * height + height * length)`.
    - `displayDimensions`: Prints the dimensions of the box.
    - Getter and Setter methods for length, width, and height: Allows for retrieving and updating the dimensions of the box.
4. **Main Method**: Demonstrates creating a `Box` object, displaying its dimensions, and calculating its volume and surface area.

Solu:

Question 7:-

Attributes:

- `totalVehicles`: Keeps track of the total number of vehicles that have passed through the toll booth.
- `totalAmountCollected`: Keeps track of the total amount of money collected by the toll booth.

Constructor:

- Initializes `totalVehicles` and `totalAmountCollected` to 0.

Methods:

- `recordVehicle(double tollAmount)`: Records a vehicle passing through and adds the toll amount to the total collected.
- `getTotalVehicles()`: Returns the total number of vehicles.
- `getTotalAmountCollected()`: Returns the total amount collected.
- `displayData()`: Displays the total number of vehicles and the total amount collected.

Main Method:

- Creates a `TollBooth` object and records a few vehicles passing through the toll booth with different toll amounts.
- Displays the toll booth data using the `displayData()` method.

Solution:

```
public class TollBooth {
  int totalVehicle;
  double AmountCollected;

  public TollBooth(){
     totalVehicle = 0;
     AmountCollected = 0.0;
  }
  public void recordVehicle(double tollAmount){
     totalVehicle++;
     AmountCollected = AmountCollected+tollAmount;
  }
  public void displayData(){
     System.out.println(totalVehicle);
     System.out.println(AmountCollected);
  }

  public static void main(String[] args) {
     TollBooth booth = new TollBooth();
     booth.recordVehicle(30.7);
     booth.recordVehicle(9.3);
```

```
        booth.recordVehicle(65.8);
        booth.displayData();
    }
}
```

Ques 8:

Static Members

- **Question**: Create a `Counter` class with a static attribute `count` and a non-static method `increment` that increments the count. Write a main method to create multiple `Counter` objects and demonstrate how the static `count` attribute is shared among them.

Solution:-

Ques 9 :

Inheritance

- **Question**: Define a `Shape` class with an attribute `colour` and a method `displayColor`. Create a subclass `Circle` that extends `Shape` and adds an attribute `radius` and a method to calculate the area. Write a main method to create a `Circle` object and demonstrate method overriding.

Ques 10 :

Composition

- **Question**: Create a `Library` class with attributes `name` and a list of `Book` objects. Add methods to add a book, remove a book, and display all books in the library. Write a main method to demonstrate the composition relationship.

Ques 11 :

Polymorphism

- **Question**: Define an interface `Animal` with a method `sound`. Create classes `Dog` and `Cat` that implement `Animal` and provide specific implementations of the `sound` method. Write a main method to demonstrate polymorphism by creating a list of `Animal` objects and calling the `sound` method on each.

Ques 12 :

Abstract Class

- **Question**: Create an abstract class `Employee` with attributes `name` and `id`, and an abstract method `calculateSalary`. Create subclasses `FullTimeEmployee` and `PartTimeEmployee` that extend `Employee` and provide specific implementations of the `calculateSalary` method. Write a main method to demonstrate the use of abstract classes.

Ques13:

Ques:-

Explanation

1. **Class Definition**: The `Marketer` class encapsulates the properties (`name`, `company`, `yearsOfExperience`, `specialisation`) and behaviours of a marketer.
2. **Constructor**: Initialises the `Marketer` object with the specified attributes.
3. **Methods**:
    - `displayDetails`: Prints the details of the marketer.
    - `promoteProduct`: Simulates promoting a product by printing a message.

- `calculateEffectiveness`: Calculates and prints the conversion rate of a marketing campaign based on leads generated and sales made.
- Getter and Setter methods for `name`, `company`, `yearsOfExperience`, and `specialisation`: Allows for retrieving and updating these attributes.

4. **Main Method**: Demonstrates creating a `Marketer` object, displaying its details, promoting a product, and calculating the effectiveness of a marketing campaign.

Solution:-

```
public class Marketer {
    private String name;
    private String company;
    private int yearsOfExperience;
    private String specialisation;

    // Constructor to initialise the attributes
    public Marketer(String name, String company, int yearsOfExperience, String
specialisation) {
        this.name = name;
        this.company = company;
        this.yearsOfExperience = yearsOfExperience;
        this.specialization = specialisation;
    }

    // Method to display the details of the marketer
    public void displayDetails() {
        System.out.println("Name: " + name);
        System.out.println("Company: " + company);
        System.out.println("Years of Experience: " + yearsOfExperience);
        System.out.println("Specialisation: " + specialisation);
    }

    // Method to promote a product
    public void promoteProduct(String product) {
        System.out.println(name + " is promoting the product: " + product);
    }

    // Method to calculate the effectiveness of a marketing campaign
    public void calculateEffectiveness(int leadsGenerated, int salesMade) {
        if (leadsGenerated > 0) {
            double conversionRate = (double) salesMade / leadsGenerated * 100;
            System.out.println("Conversion Rate: " + String.format("%.2f", conversionRate) +
"%");
        } else {
```

```
            System.out.println("No leads generated, effectiveness cannot be calculated.");
        }
    }

    // Getter and Setter methods for name
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // Getter and Setter methods for company
    public String getCompany() {
        return company;
    }

    public void setCompany(String company) {
        this.company = company;
    }

    // Getter and Setter methods for yearsOfExperience
    public int getYearsOfExperience() {
        return yearsOfExperience;
    }

    public void setYearsOfExperience(int yearsOfExperience) {
        this.yearsOfExperience = yearsOfExperience;
    }

    // Getter and Setter methods for specialisation
    public String getSpecialization() {
        return specialisation;
    }

    public void setSpecialization(String specialisation) {
        this.specialization = specialisation;
    }

    public static void main(String[] args) {
        // Create a new Marketer object
        Marketer marketer = new Marketer("Alice Smith", "Tech Innovators", 5, "Digital
Marketing");

        // Display the details of the marketer
        marketer.displayDetails();
```

```
        // Promote a product
        marketer.promoteProduct("Innovative Tech Gadget");

        // Calculate the effectiveness of a marketing campaign
        marketer.calculateEffectiveness(1000, 250);
    }
}
```

Ques14:

Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.

```
{{1, 2, 3},

{4, 0, 6},

{7, 8, 9}}

Solutions;
```

```
import java.util.Arrays;

class SetMatrixZeroes {
    public void setZeroes(int[][] matrix){
        int n = matrix.length;
        int m = matrix[0].length;
        int[] rowArray = new int[n];
        int[] colArray = new int[m];

        Arrays.fill(rowArray,1);
        Arrays.fill(colArray,1);

        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(matrix[i][j]==0){
                    rowArray[i]=0;
                    colArray[j]=0;
                }
            }
        }
        for(int i=0;i<n;i++){
            for (int j=0;j<m;j++){
                if(rowArray[i]==0 || colArray[j]==0){
```

```
                matrix[i][j]=0;
            }
        }
    }
  }
}

public class Main {
  public static void main(String[] args) {
    int[][] matrix = {
            {1, 2, 3},
            {4, 0, 6},
            {7, 8, 9}
    };

    SetMatrixZeroes smz = new SetMatrixZeroes();
    smz.setZeroes(matrix);

    for (int i = 0; i < matrix.length; i++) {
      for (int j = 0; j < matrix[0].length; j++) {
        System.out.print(matrix[i][j] + " ");
      }
      System.out.println();
    }
  }
}
```

Ques 15:

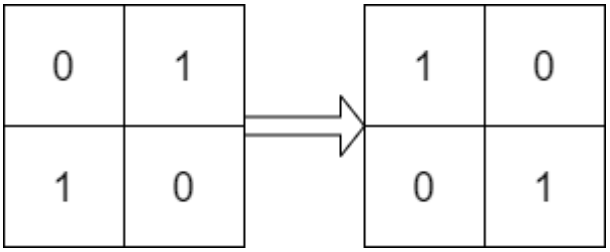
Rotate a matrix by 90 degree in clockwise direction without using any extra space

Input:
1 2 3
4 5 6
7 8 9

Output:
7 4 1
8 5 2
9 6 3
Solution

Q16:

Given two n x n binary matrices mat and target, return true *if it is possible to make* mat *equal to* target *by* ***rotating*** mat *in* ***90-degree increments****, or* false *otherwise.*



Solution:

```
class Solution {
  public boolean findRotation(int[][] matrix, int[][] target) {

     //for comparing the matrix till 4 rotations i.e., 0,90,180,270 degrees
     for(int i = 0; i < 4; i++){
        if(Arrays.deepEquals(matrix,target)) return true;
        matrix_90D(matrix);
     }
     return false;
  }

  //for rotating the matrix by 90 degree:
  public static void matrix_90D(int[][] matrix){
     int n = matrix.length;
     for(int i = 0 ; i < n-1 ; i++){
        for(int j = i+1 ; j < n ; j++){
           int temp = matrix[i][j];
           matrix[i][j] = matrix[j][i];
           matrix[j][i] = temp;
        }
     }

     for(int i = 0 ; i < n ; i++){
        for(int j = 0 ; j < n/2 ; j++){
           int temp = matrix[i][j];
           matrix[i][j] = matrix[i][n-1-j];
           matrix[i][n-1-j] = temp;
        }
     }
  }
```

```
}
```

Ques17:

Initially, there is a grid with some cells which may be alive or dead. Our task is to generate the next generation of cells based on the following rules:

Any live cell with fewer than two live neighbours dies as if caused by underpopulation.

Any live cell with two or three live neighbours lives on to the next generation.

Any live cell with more than three live neighbours dies, as if by overpopulation.

Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

Input :

1, 0, 1, 0,1

0,1, 0, 1, 0

1, 0, 0, 1,0

0, 0,1 ,1, 0

1, 1, 0, 0, 1

Output:

0, 1, 0, 1, 0

1, 0 ,1, 0, 0

0, 0, 0, 1, 0

0, 0, 0, 1, 0

0, 0, 0, 0, 0

Solution

```
package Day4;
import java.util.Scanner;
public class DeadOrAlive {
   public static void main(String[] args) {
      Scanner sc= new Scanner(System.in);
      int n = sc.nextInt();
      int m = sc.nextInt();
      int[][] mat = new int[n][m];
      for(int i=0;i<n;i++){
         for(int j=0;j<m;j++){
            mat[i][j] = sc.nextInt();
         }
      }
      int[][] res = new int [n][m];
      for(int i=0;i<n;i++){
         for(int j=0;j<m;j++){
            int live = live(i,j,mat);
            if(mat[i][j]==1 && live<2){
               res[i][j]=0;
            }else if(mat[i][j]==1 && (live==2 || live==3)){
               res[i][j]=1;
            }else if(mat[i][j]==1 && live>3){
               res[i][j]=0;
            }else if(mat[i][j]==0 && live==3){
               res[i][j]=1;
            }
         }
      }
      for(int i=0;i<n;i++){
         for(int j=0;j<m;j++){
            System.out.print(res[i][j] + " ");
         }
         System.out.println();
      }
   }
   public static int live(int i,int j,int[][] mat){
      int c=0;
      if(i-1>=0 && mat[i-1][j]==1) c++;     //top
```

```
      if(i+1< mat.length && mat[i+1][j]==1 ) c++;   //bottom
      if(j-1>=0 && mat[i][j-1]==1) c++; //left
      if(j+1<mat[0].length && mat[i][j+1]==1) c++; //right
      return c;
   }
}
```

Ques18:-

Given an m x n matrix, return true if the matrix is Toeplitz. Otherwise, return false.
A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.

1	2	3	4
5	1	2	3
9	5	1	2

Explanation:

In the above grid, the diagonals are:
"[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]".
In each diagonal all elements are the same, so the answer is True.

19.Happy Number

Problem Statement :

Write an algorithm to determine if a number n is happy.
A happy number is a number defined by the following process:

- Starting with any positive integer, replace the number by the sum of the squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.
- Those numbers for which this process ends in 1 are happy.

Return true if n is a happy number, and false if not.

Example 1:
Input: n = 19
Output: true

Explanation:

$1^2 + 9^2 = 82$
$8^2 + 2^2 = 68$
$6^2 + 8^2 = 100$
$1^2 + 0^2 + 0^2 = 1$

Example 2:
Input: n = 2
Output: false

Constraints:

$1 <= n <= 2^{31} - 1$

Q20.Print matrix in snake pattern

Given an n x n matrix. In the given matrix, you have to print the elements of the matrix in the snake pattern.

Input: *mat[][] = {{10, 20, 30, 40},*
{15, 25, 35, 45},
{27, 29, 37, 48},
{32, 33, 39, 50}}

Output: *10 20 30 40 45 35 25 15 27 29 37 48 50 39 33 32*

Solution:

```
package Day5;

class SnakesPattern{
   static void print(int[][] mat) {
      for (int i=0;i<mat.length;i++){
         if(i%2 == 0){
            for(int j=0;j<mat[0].length;j++)
               System.out.print(mat[i][j]+" ");
         }
         else{
            for(int j=mat[0].length-1;j>=0;j--)
               System.out.print(mat[i][j]+" ");
         }
      }
   }
   public static void main(String[] args) {
      int mat[][]=new int[][]{
            {10,20,30,40},
            {15,25,35,45},
            {27,29,37,48},
            {32,33,39,50} };
      print(mat);
   }
}
```