Ques 1 :-

Sherlock has a matrix of dimension [nxm]. All the elements in this matrix are zero (Initialised as 0).
Sherlock has q queries to execute on the matrix. Each query has two integers i.e, the type of query and the index.
So if the type is 0, the index represents the index of the row. Similarly, if the type is 1, the index represents the index of the column.
Sherlock has to increase the value by 1 of all the elements that are present in that row or column depending on the type and index.
In the end, after executing all queries, Sherlock is curious and wants to find the number of elements of the matrix which are odd (not divisible by 2).
Note - 0-based indexing is followed here.
input
The first line of the input contains one integer t (1 st≤ 10) - the number of test cases. Then t test cases follow.
The first line of each test case contains two integers n,m (1≤ n ≤ 1000, 1< m ≤1000) - the number of rows and columns, respectively.
The second line of each test case contains a single integer q (1 ≤qs 100000) - the number of queries.
The next q lines of each test case contain 2 integers type and index (0 ≤ type ≤ 1, 0 ≤ index ≤ n-1 (for row) and m-1 (for col)) - the info on the query.

Input:

Test case 1:
2 2
4
0 1
0 0
0 0
1 1

Test case 2:

```
4 4
10
0 1
0 2
0 0
0 1
0 1
0 1
1 1
1 1
1 3
1 0
```

Output 2 ,8

Solution:

```
import java.util.*;
public class Main{
    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];
        int q = sc.nextInt();
        for(int i=1;i<=q;i++){
            int a = sc.nextInt();
            int b = sc.nextInt();
            solve(mat,n,m,a,b);
        }
        int count =0;
        for(int i =0;i<n;i++){
```

```
        for(int j=0;j<m;j++){
            if(mat[i][j]%2!=0){
                count++;
            }
        }
    }
    System.out.println("Answer is :" + count);
}
public static void solve(int[][] mat,int n,int m,int a,int b){
    if (a==0){
        for(int i=0;i<m;i++){
            mat[b][i]+=1;
        }
    }
    else{
        for(int i=0;i<n;i++){
            mat[i][b]+=1;
        }
    }
}
}
```

Ques 2: Find the sum of diagonals

{{1, 2, 3,4},

{5, 6, 7,8},
{9, 10,11,12},
{13,14,15,16}}

Q: - Multiplication of two matrix

int[][] a = new int[][]{{1, 4, 2}, {2, 5, 3}, {3, 6, 1}};
int[][] b = new int[][]{{2, 3, 2}, {1, 1, 5}, {4, 8, 6}};

Solution:

```
public class Main {
        public static void main(String[] args) {

     int[][] a = new int[][]{{1, 4, 2}, {2, 5, 3}, {3, 6, 1}};
     int[][] b = new int[][]{{2, 3, 2}, {1, 1, 5}, {4, 8, 6}};

     multiplyMatrix(a,b);



  }

  public static void multiplyMatrix(int[][] a,int[][] b){
     int ans[][]=new int[a.length][a[0].length];

     for (int i = 0; i < a.length; i++) {
        for(int j=0;j<a[0].length;j++){
           int sum=0;
           for(int k=0;k<b.length;k++){
              sum+=a[i][k]*b[k][j];
           }
           ans[i][j]=sum;
        }
     }

     for(int[] x:ans){
        for(int y:x){
           System.out.print(y+" ");
        }
        System.out.println();
     }
```

```
   }
}
```

Q: Spiral Matrix
{{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}}

OutPut:

```
1
2
3
4
8
12
16
15
14
13
9
5
6
7
11
10
```

Solution:

```
class Main{
   public static void spiralMatrix(int[][] matrix){
       int startRow = 0;
       int startCol = 0;
       int endRow = matrix.length-1;
       int endCol = matrix[0].length-1;
       while(startRow<=endRow && startCol<=endCol){
//            top
           for(int j=startCol;j<=endCol;j++){
               System.out.println(matrix[startRow][j] + " ");
           }
//            right
           for(int i=startRow+1;i<=endRow;i++){
               System.out.println(matrix[i][endCol] + "");
           }
//            bottom
           for(int j =endCol-1;j>=startCol;j--){
               System.out.println(matrix[endRow][j] + " ");
           }
//            left
           for(int i = endRow-1;i>=startRow+1;i--){
               System.out.println(matrix[i][startCol] + " ");
           }
           startRow++;
           endRow--;
           startCol++;
           endCol--;
       }
       System.out.println();

   }

   public static void main(String[] args) {
       int[][] matrix = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
       spiralMatrix(matrix);
   }
}
```

Ques:-
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.

Question:

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 initialize 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 initialize 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 initialize 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.

Solution :

```
import java.util.ArrayList;

class Product {
    private int productId;
    private String productName;
    private double price;

    public Product(int productId, String productName, double price) {
        this.productId = productId;
        this.productName = productName;
        this.price = price;
    }

    public int getProductId() {
        return productId;
    }

    public void setProductId(int productId) {
        this.productId = productId;
    }

    public String getProductName() {
        return productName;
    }

    public void setProductName(String productName) {
        this.productName = productName;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public void displayProductDetails() {
        System.out.println("Product ID: " + productId);
        System.out.println("Product Name: " + productName);
        System.out.println("Price: $" + price);
    }
}

class Electronics extends Product {
    private String brand;
    private int warranty;

    public Electronics(int productId, String productName, double price, String brand, int
warranty) {
        super(productId, productName, price);
        this.brand = brand;
        this.warranty = warranty;
    }

    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public int getWarranty() {
        return warranty;
    }

    public void setWarranty(int warranty) {
        this.warranty = warranty;
    }

    @Override
    public void displayProductDetails() {
        super.displayProductDetails();
        System.out.println("Brand: " + brand);
        System.out.println("Warranty: " + warranty + " months");
    }
}

class Clothing extends Product {
    private String size;
    private String material;
    public Clothing(int productId, String productName, double price, String size, String
material) {
        super(productId, productName, price);
        this.size = size;
        this.material = material;
    }

    public String getSize() {
        return size;
    }

    public void setSize(String size) {
        this.size = size;
    }

    public String getMaterial() {
        return material;
    }

    public void setMaterial(String material) {
        this.material = material;
    }

    @Override
    public void displayProductDetails() {
        super.displayProductDetails();
        System.out.println("Size: " + size);
        System.out.println("Material: " + material);
    }
}

class ShoppingCart {
    private ArrayList<Product> cartItems;

    public ShoppingCart() {
        this.cartItems = new ArrayList<>();
    }

    public void addProduct(Product product) {
        cartItems.add(product);
    }

    public void removeProduct(int productId) {
        cartItems.removeIf(product -> product.getProductId() == productId);
    }

    public void displayCartDetails() {
        for (Product product : cartItems) {
            product.displayProductDetails();
            System.out.println();
        }
    }

    public double calculateTotal() {
        double total = 0;
        for (Product product : cartItems) {
            total += product.getPrice();
        }
        return total;
    }

    public static void main(String[] args) {
        // Create instances of Electronics and Clothing
        Electronics laptop = new Electronics(1, "Laptop", 1000.0, "Dell", 24);
        Clothing tshirt = new Clothing(2, "T-Shirt", 20.0, "L", "Cotton");

        // Create a ShoppingCart and add products
        ShoppingCart cart = new ShoppingCart();
        cart.addProduct(laptop);
        cart.addProduct(tshirt);

        // Display cart details
        System.out.println("Cart Details:");
        cart.displayCartDetails();

        // Calculate and display total price
        double totalPrice = cart.calculateTotal();
        System.out.println("Total Price: $" + totalPrice);

        // Remove a product and display updated cart details
        cart.removeProduct(1);
        System.out.println("Updated Cart Details:");
        cart.displayCartDetails();
    }
}
```

Question:

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.

CODE:
```
class Book {
    // Properties
    private String title;
    private String author;
    private double price;

    // Constructor
    public Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    // Getter for title
    public String getTitle() {
        return title;
    }

    // Setter for title
    public void setTitle(String title) {
        this.title = title;
    }
    // Getter for author
    public String getAuthor() {
        return author;
    }

    // Setter for author
    public void setAuthor(String author) {
        this.author = author;
    }

    // Getter for price
    public double getPrice() {
        return price;
    }

    // Setter for price
    public void setPrice(double price) {
        this.price = price;
    }

    // Method to display book details
    public void displayDetails() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Price: $" + price);
    }
}

public class Main {
    public static void main(String[] args) {
        // Create an instance of Book
        Book myBook = new Book("The Great Gatsby", "F. Scott Fitzgerald", 10.99);

        // Set properties using setters
        myBook.setTitle("The Great Gatsby");
        myBook.setAuthor("F. Scott Fitzgerald");
        myBook.setPrice(10.99);

        // Display book details
        myBook.displayDetails();
    }
}
```

Ques:

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.

Code:

```
class Student {
    // Properties
    private String name;
    private int rollNumber;
    private double marks;

    // Constructor
    public Student(String name, int rollNumber, double marks) {
        this.name = name;
        this.rollNumber = rollNumber;
        this.marks = marks;
    }

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

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

    // 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
    public void displayDetails() {
        System.out.println("Name: " + name);
        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("John Doe", 101, 85.5);

      // Set properties using setters
      student.setName("John Doe");
      student.setRollNumber(101);
      student.setMarks(85.5);

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

Question3 :

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 initialize 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 initialize 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.

Code:

```
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("John Doe", 20, 101, 85.5);

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

Question:-
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

```
// Class representing a toll booth
public class TollBooth {
   // Attributes
   private int totalVehicles;
   private double totalAmountCollected;

   // Constructor
   public TollBooth() {
      totalVehicles = 0;
      totalAmountCollected = 0.0;
   }

   // Method to record a vehicle passing through the toll booth
   public void recordVehicle(double tollAmount) {
      totalVehicles++;
      totalAmountCollected += tollAmount;
   }

   // Method to get the total number of vehicles
   public int getTotalVehicles() {
      return totalVehicles;
   }

   // Method to get the total amount collected
   public double getTotalAmountCollected() {
      return totalAmountCollected;
   }

   // Method to display the toll booth data
   public void displayData() {
      System.out.println("Total Vehicles: " + totalVehicles);
      System.out.println("Total Amount Collected: $" + totalAmountCollected);
   }

   // Main method for testing
   public static void main(String[] args) {
      TollBooth booth = new TollBooth();
      booth.recordVehicle(2.50);
      booth.recordVehicle(3.00);
      booth.recordVehicle(4.75);
      booth.displayData();
   }
}
```

Ques:-

Explanation

1. **Class Definition**: The `Box` class encapsulates the properties (length, width, height) and behaviors of a box.
2. **Constructor**: Initializes 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.

Solution:-

```
public class Box {
    private double length;
    private double width;
    private double height;

    // Constructor to initialise the dimensions of the box
    public Box(double length, double width, double height) {
        this.length = length;
        this.width = width;
        this.height = height;
    }

    // Method to calculate the volume of the box
    public double calculateVolume() {
        return length * width * height;
    }
```

```
// Method to calculate the surface area of the box
public double calculateSurfaceArea() {
   return 2 * (length * width + width * height + height * length);
}

// Method to display the dimensions of the box
public void displayDimensions() {
   System.out.println("Length: " + length);
   System.out.println("Width: " + width);
   System.out.println("Height: " + height);
}

// Getter and Setter methods for length
public double getLength() {
   return length;
}

public void setLength(double length) {
   this.length = length;
}

// Getter and Setter methods for width
public double getWidth() {
   return width;
}

public void setWidth(double width) {
   this.width = width;
}

// Getter and Setter methods for height
public double getHeight() {
   return height;
}

public void setHeight(double height) {
   this.height = height;
}

public static void main(String[] args) {
   // Create a new Box object
   Box myBox = new Box(10.0, 5.0, 7.0);

   // Display the dimensions of the box
   myBox.displayDimensions();

   // Calculate and display the volume of the box
   double volume = myBox.calculateVolume();
```

```
        System.out.println("Volume: " + volume);

        // Calculate and display the surface area of the box
        double surfaceArea = myBox.calculateSurfaceArea();
        System.out.println("Surface Area: " + surfaceArea);
    }
}
```

Ques-

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.**

Soluti0n:

```
public class BookCD {
    private String title;
    private String authorOrArtist;
    private double price;
    private String type; // "Book" or "CD"

    // Constructor to initialise the attributes
    public BookCD(String title, String authorOrArtist, double price, String type) {
        this.title = title;
        this.authorOrArtist = authorOrArtist;
        this.price = price;
        this.type = type;
    }
```

```
}

// Method to display the details of the Book or CD
public void displayDetails() {
    System.out.println("Type: " + type);
    System.out.println("Title: " + title);
    System.out.println("Author/Artist: " + authorOrArtist);
    System.out.println("Price: $" + price);
}

// Method to apply a discount to the price
public void applyDiscount(double discountPercentage) {
    if (discountPercentage > 0 && discountPercentage <= 100) {
        price -= price * (discountPercentage / 100);
    } else {
        System.out.println("Invalid discount percentage!");
    }
}

// Getter and Setter methods for title
public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

// Getter and Setter methods for authorOrArtist
public String getAuthorOrArtist() {
    return authorOrArtist;
}

public void setAuthorOrArtist(String authorOrArtist) {
    this.authorOrArtist = authorOrArtist;
}

// Getter and Setter methods for price
public double getPrice() {
    return price;
}

public void setPrice(double price) {
    this.price = price;
}

// Getter and Setter methods for type
public String getType() {
    return type;
    }

    public void setType(String type) {
        this.type = type;
    }

    public static void main(String[] args) {
        // Create a new BookCD object for a Book
        BookCD book = new BookCD("The Great Gatsby", "F. Scott Fitzgerald", 10.99,
"Book");

        // Display the details of the book
        book.displayDetails();

        // Apply a discount to the book
        book.applyDiscount(10);
        System.out.println("Price after discount: $" + book.getPrice());

        // Create a new BookCD object for a CD
        BookCD cd = new BookCD("Thriller", "Michael Jackson", 14.99, "CD");

        // Display the details of the CD
        cd.displayDetails();

        // Apply a discount to the CD
        cd.applyDiscount(15);
        System.out.println("Price after discount: $" + cd.getPrice());
    }
}
```

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 specialization;

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

    // 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("Specialization: " + specialization);
    }

    // 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);
    }
}
```

Ques 9 :

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.

```
public class Counter {
    // Static attribute to keep track of the count
    private static int count = 0;

    // Non-static method to increment the count
    public void increment() {
        count++;
    }

    // Static method to get the current count
    public static int getCount() {
        return count;
    }

    // Main method to demonstrate the functionality
    public static void main(String[] args) {
        Counter counter1 = new Counter();
        Counter counter2 = new Counter();
        Counter counter3 = new Counter();

        counter1.increment();
        System.out.println("Counter1 incremented count: " + Counter.getCount()); // Output: 1

        counter2.increment();
        System.out.println("Counter2 incremented count: " + Counter.getCount()); // Output: 2

        counter3.increment();
        System.out.println("Counter3 incremented count: " + Counter.getCount()); // Output: 3
        // Demonstrating that all Counter objects share the same static count
        System.out.println("Final count from Counter1: " + counter1.getCount()); // Output: 3
        System.out.println("Final count from Counter2: " + counter2.getCount()); // Output: 3
        System.out.println("Final count from Counter3: " + counter3.getCount()); // Output: 3
    }
}
```

Ques 10 :

Inheritance

- **Question**: Define a `Shape` class with an attribute `color` 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.

```
public class Shape {
    // Attribute
    protected String color;

    // Constructor
    public Shape(String color) {
        this.color = color;
    }

    // Method to display the color
    public void displayColor() {
        System.out.println("Color: " + color);
    }
}
public class Circle extends Shape {
    // Attribute
    private double radius;

    // Constructor
    public Circle(String colour, double radius) {
        super(colour); // Call the constructor of the superclass (Shape)
        this.radius = radius;
    }
```

```
 // Method to calculate the area
    public double calculateArea() {
        return Math.PI * radius * radius;
    }

    // Overriding the displayColor method
    @Override
    public void displayColor() {
        System.out.println("Circle Color: " + color);
    }

    // Method to display the radius and area
    public void displayDetails() {
        displayColor();
        System.out.println("Radius: " + radius);
        System.out.println("Area: " + calculateArea());
    }

    // Main method to demonstrate the functionality
    public static void main(String[] args) {
        Circle circle = new Circle("Red", 5.0);
        circle.displayDetails();
    }
}
```

Ques 11 :

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
-

```
import java.util.Objects;

public class Book {
    // Attributes
    private String title;
    private String author;

    // Constructor
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    // Getters
    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    // toString method to display book details
    @Override
    public String toString() {
        return "Title: " + title + ", Author: " + author;
    }

    // Equals and hashCode methods for comparison in lists
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Book book = (Book) o;
        return Objects.equals(title, book.title) && Objects.equals(author, book.author);
    }

    @Override
    public int hashCode() {
        return Objects.hash(title, author);
    }
}
import java.util.ArrayList;
import java.util.List;

public class Library {
    // Attributes
    private String name;
    private List<Book> books;

    // Constructor
    public Library(String name) {
        this.name = name;
        this.books = new ArrayList<>();
    }

    // Method to add a book to the library
    public void addBook(Book book) {
        books.add(book);
        System.out.println(book.getTitle() + " by " + book.getAuthor() + " added to the
library.");
    }

    // Method to remove a book from the library
    public void removeBook(Book book) {
        if (books.remove(book)) {
            System.out.println(book.getTitle() + " by " + book.getAuthor() + " removed from the
library.");
        } else {
            System.out.println(book.getTitle() + " by " + book.getAuthor() + " not found in the
library.");
        }
    }

    // Method to display all books in the library
    public void displayBooks() {
        if (books.isEmpty()) {
            System.out.println("The library is empty.");
        } else {
            System.out.println("Books in " + name + " Library:");
            for (Book book : books) {
                System.out.println(book);
            }
        }
    }

    // Main method to demonstrate the functionality
    public static void main(String[] args) {
        // Create a Library object
        Library library = new Library("City Library");

        // Create some Book objects
        Book book1 = new Book("To Kill a Mockingbird", "Harper Lee");
        Book book2 = new Book("1984", "George Orwell");
        Book book3 = new Book("The Great Gatsby", "F. Scott Fitzgerald");

        // Add books to the library
        library.addBook(book1);
        library.addBook(book2);
        library.addBook(book3);

        // Display all books in the library
        library.displayBooks();

        // Remove a book from the library
        library.removeBook(book2);
        // Display all books in the library after removal
        library.displayBooks();
    }
}
```

Ques 12 :

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.

```
// Define the Animal interface
public interface Animal {
    // Abstract method to be implemented by classes that implement this interface
    void sound();
}
// Implement the Animal interface in the Dog class
public class Dog implements Animal {
    @Override
    public void sound() {
        System.out.println("Dog barks");
    }
}
// Implement the Animal interface in the Cat class
public class Cat implements Animal {
    @Override
    public void sound() {
        System.out.println("Cat meows");
    }
}
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        // Create a list of Animal objects
        List<Animal> animals = new ArrayList<>();
        // Add Dog and Cat objects to the list
        animals.add(new Dog());
        animals.add(new Cat());

        // Call the sound method on each Animal object
        for (Animal animal : animals) {
            animal.sound();
        }
    }
}
```

Ques 13 :

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.

```
// Define the abstract Employee class
public abstract class Employee {
    // Attributes
    private String name;
    private int id;

    // Constructor
    public Employee(String name, int id) {
        this.name = name;
        this.id = id;
    }

    // Abstract method to be implemented by subclasses
    public abstract double calculateSalary();

    // Getters and setters
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
}
// Define the FullTimeEmployee class that extends Employee
public class FullTimeEmployee extends Employee {
    private double annualSalary;

    // Constructor
    public FullTimeEmployee(String name, int id, double annualSalary) {
        super(name, id);
        this.annualSalary = annualSalary;
    }

    // Implementation of the abstract method
    @Override
    public double calculateSalary() {
        return annualSalary;
    }

    // Getter and setter for annualSalary
    public double getAnnualSalary() {
        return annualSalary;
    }

    public void setAnnualSalary(double annualSalary) {
        this.annualSalary = annualSalary;
    }
}

// Define the PartTimeEmployee class that extends Employee
public class PartTimeEmployee extends Employee {
    private double hourlyWage;
    private int hoursWorked;

    // Constructor
    public PartTimeEmployee(String name, int id, double hourlyWage, int hoursWorked) {
        super(name, id);
        this.hourlyWage = hourlyWage;
        this.hoursWorked = hoursWorked;
    }

    // Implementation of the abstract method
    @Override
    public double calculateSalary() {
        return hourlyWage * hoursWorked;
    }

    // Getters and setters
    public double getHourlyWage() {
        return hourlyWage;
    }

    public void setHourlyWage(double hourlyWage) {
        this.hourlyWage = hourlyWage;
    }

    public int getHoursWorked() {
        return hoursWorked;
    }

    public void setHoursWorked(int hoursWorked) {
        this.hoursWorked = hoursWorked;
    }
}

public class Main {
    public static void main(String[] args) {
        // Create FullTimeEmployee and PartTimeEmployee objects
        Employee fullTimeEmployee = new FullTimeEmployee("Alice", 1, 60000);
        Employee partTimeEmployee = new PartTimeEmployee("Bob", 2, 20, 120);

        // Display the salary of each employee
        System.out.println(fullTimeEmployee.getName() + "'s salary: $" +
fullTimeEmployee.calculateSalary());
        System.out.println(partTimeEmployee.getName() + "'s salary: $" +
partTimeEmployee.calculateSalary());
    }
}
```