1. a. Write a Java program to read 5 subject marks of a student and calculate the total and grade. The grade system is as follows. Letter Grade Grade Points Marks Range O (Outstanding) 10 91 – 100 A+ (Excellent) 9 81 – 90 A (Very Good) 8 71 – 80 B+ (Good) 7 61 – 70 B (Average) 6 50 – 60 RA 0 < 50

```
import java.util.Scanner;

public class StudentGradeCalculator {

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

        System.out.println("Enter marks for 5 subjects:");

        // Array to store subject marks
        int[] marks = new int[5];

        // Reading marks for each subject
        for (int i = 0; i < 5; i++) {
            System.out.print("Subject " + (i + 1) + ": ");
            marks[i] = scanner.nextInt();
        }

        // Calculate total marks
        int totalMarks = calculateTotalMarks(marks);

        // Calculate grade based on total marks
        String grade = calculateGrade(totalMarks);

        // Display results
        System.out.println("Total Marks: " + totalMarks);
```

```
        System.out.println("Grade: " + grade);
    }

    // Method to calculate total marks
    private static int calculateTotalMarks(int[] marks) {
        int total = 0;
        for (int mark : marks) {
            total += mark;
        }
        return total;
    }

    // Method to calculate grade based on total marks
    private static String calculateGrade(int totalMarks) {
        if (totalMarks >= 91 && totalMarks <= 100) {
            return "O (Outstanding)";
        } else if (totalMarks >= 81 && totalMarks <= 90) {
            return "A+ (Excellent)";
        } else if (totalMarks >= 71 && totalMarks <= 80) {
            return "A (Very Good)";
        } else if (totalMarks >= 61 && totalMarks <= 70) {
            return "B+ (Good)";
        } else if (totalMarks >= 50 && totalMarks <= 60) {
            return "B (Average)";
        } else {
            return "RA (Failed)";
        }
    }
}
```

b. Write a program that allows a user to enter three words, and display the appropriate threeletter acronym in all uppercase letters. If the user enters more than three words, ignore the extra words.

```
import java.util.Scanner;

public class AcronymGenerator {

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

        System.out.println("Enter three words:");

        // Reading three words from the user
        String word1 = scanner.next();
        String word2 = scanner.next();
        String word3 = scanner.next();

        // Generating the three-letter acronym
        String acronym = generateAcronym(word1, word2, word3);

        // Displaying the acronym in uppercase
        System.out.println("Acronym: " + acronym.toUpperCase());
    }

    // Method to generate three-letter acronym
    private static String generateAcronym(String word1, String word2, String word3) {
        // Taking the first letter from each word
        char letter1 = word1.charAt(0);
        char letter2 = word2.charAt(0);
```

```
        char letter3 = word3.charAt(0);

        // Concatenating the letters to form the acronym
        return "" + letter1 + letter2 + letter3;
    }
}
```

2. a. Define a class named COMPLEX for representing complex numbers that contains necessary data members and member functions. A complex number has the general form a + ib, where a is the real part and b is the imaginary part (i stands for imaginary). Include methods for all the four basic arithmetic operators.

```
public class Complex {
    private double real;
    private double imaginary;

    // Constructor
    public Complex(double real, double imaginary) {
        this.real = real;
        this.imaginary = imaginary;
    }

    // Getter methods
    public double getReal() {
        return real;
    }

    public double getImaginary() {
        return imaginary;
    }

    // Method to add two complex numbers
```

```
public Complex add(Complex other) {
    double newReal = this.real + other.real;
    double newImaginary = this.imaginary + other.imaginary;
    return new Complex(newReal, newImaginary);
}

// Method to subtract two complex numbers
public Complex subtract(Complex other) {
    double newReal = this.real - other.real;
    double newImaginary = this.imaginary - other.imaginary;
    return new Complex(newReal, newImaginary);
}

// Method to multiply two complex numbers
public Complex multiply(Complex other) {
    double newReal = this.real * other.real - this.imaginary * other.imaginary;
    double newImaginary = this.real * other.imaginary + this.imaginary * other.real;
    return new Complex(newReal, newImaginary);
}

// Method to divide two complex numbers
public Complex divide(Complex other) {
    double denominator = other.real * other.real + other.imaginary * other.imaginary;
    double newReal = (this.real * other.real + this.imaginary * other.imaginary) / denominator;
    double newImaginary = (this.imaginary * other.real - this.real * other.imaginary) / denominator;
    return new Complex(newReal, newImaginary);
}

// Method to display the complex number
```

```
    public void display() {
        System.out.println(real + " + " + imaginary + "i");
    }

    public static void main(String[] args) {
        Complex complex1 = new Complex(2, 3);
        Complex complex2 = new Complex(1, -1);

        // Testing arithmetic operations
        Complex sum = complex1.add(complex2);
        Complex difference = complex1.subtract(complex2);
        Complex product = complex1.multiply(complex2);
        Complex quotient = complex1.divide(complex2);

        // Displaying results
        System.out.print("Sum: ");
        sum.display();

        System.out.print("Difference: ");
        difference.display();

        System.out.print("Product: ");
        product.display();

        System.out.print("Quotient: ");
        quotient.display();
    }
}
```

b. Write a Java program that determines the number of days in a month.

```
import java.util.Scanner;

public class DaysInMonth {
    public static void main(String[] args) {
        // Create a Scanner object for user input
        Scanner scanner = new Scanner(System.in);

        // Prompt user for input
        System.out.print("Enter the year: ");
        int year = scanner.nextInt();

        System.out.print("Enter the month (1-12): ");
        int month = scanner.nextInt();

        // Close the Scanner to avoid resource leak
        scanner.close();

        // Determine the number of days in the given month
        int daysInMonth = getDaysInMonth(year, month);

        // Display the result
        if (daysInMonth != -1) {
            System.out.println("Number of days in the specified month: " + daysInMonth);
        } else {
            System.out.println("Invalid input. Please enter a valid month (1-12).");
        }
    }
```

```
    // Method to determine the number of days in a month
    public static int getDaysInMonth(int year, int month) {
        if (month < 1 || month > 12) {
            return -1; // Invalid month
        }

        switch (month) {
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
            case 2:
                if (isLeapYear(year)) {
                    return 29;
                } else {
                    return 28;
                }
            default:
                return 31;
        }
    }

    // Method to check if a year is a leap year
    public static boolean isLeapYear(int year) {
        return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
    }
}
```

3.. Write a program that inserts parentheses, a space, and a dash into a string of 10 user-entered numbers to format it as a phone number. For example, 5153458912 becomes (515) 345-8912. If the user does not enter exactly 10 digits, display an error message. Continue to accept user input until the user enters 999. Save the file as PhoneNumberFormat.java.

```
import java.util.Scanner;


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


      while (true) {
         System.out.print("Enter a 10-digit number (or 999 to exit): ");
         String input = scanner.nextLine();


         if (input.equals("999")) {
            System.out.println("Exiting program.");
            break;
         }


         if (isValidPhoneNumber(input)) {
            String formattedNumber = formatPhoneNumber(input);
            System.out.println("Formatted phone number: " + formattedNumber);
         } else {
            System.out.println("Error: Please enter exactly 10 digits.");
         }
      }


      scanner.close();
   }
```

```
    private static boolean isValidPhoneNumber(String input) {
        // Check if the input consists of exactly 10 digits
        return input.matches("\\d{10}");
    }

    private static String formatPhoneNumber(String input) {
        // Format the phone number as (XXX) XXX-XXXX
        return "(" + input.substring(0, 3) + ") " + input.substring(3, 6) + "-" + input.substring(6);
    }
}
```

b. Write a Java program to calculate the revenue from a sale based on the unit price and quantity of a product input by the user. The discount rate is 10% for the quantity purchased between 100 and 120 units, and 15% for the quantity purchased greater than 120 units. If the quantity purchased is less than 100 units, the discount rate is 0%.

```
import java.util.Scanner;

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

        System.out.print("Enter the unit price: ");
        double unitPrice = scanner.nextDouble();

        System.out.print("Enter the quantity: ");
        int quantity = scanner.nextInt();

        double discountRate = calculateDiscountRate(quantity);
        double totalRevenue = calculateRevenue(unitPrice, quantity, discountRate);
```

```
        System.out.println("Discount rate: " + (discountRate * 100) + "%");
        System.out.println("Total revenue: $" + totalRevenue);

        scanner.close();
    }

    private static double calculateDiscountRate(int quantity) {
        if (quantity < 100) {
            return 0.0;
        } else if (quantity >= 100 && quantity <= 120) {
            return 0.10;
        } else {
            return 0.15;
        }
    }

    private static double calculateRevenue(double unitPrice, int quantity, double discountRate) {
        double discountedPrice = unitPrice * (1 - discountRate);
        return discountedPrice * quantity;
    }
}
```

4. a.Write an application that prompts a user for a full name and street address and constructs an ID from the user's initials and numeric part of the address. For example, the user William Henry Harrison who lives at 34 Elm would have an ID of WHH34, whereas user Addison Mitchell who lives at 1778 Monroe would have an ID of AM1778.

```
import java.util.Scanner;

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

```
    Scanner scanner = new Scanner(System.in);

    // Prompt the user for full name and street address
    System.out.print("Enter your full name: ");
    String fullName = scanner.nextLine();

    System.out.print("Enter your street address: ");
    String address = scanner.nextLine();

    // Generate the user ID
    String userID = generateUserID(fullName, address);

    // Display the generated user ID
    System.out.println("Your generated user ID is: " + userID);

    scanner.close();
}

private static String generateUserID(String fullName, String address) {
    // Extract initials from the full name
    String initials = getInitials(fullName);

    // Extract numeric part from the address
    String numericPart = getNumericPart(address);

    // Construct the user ID by combining initials and numeric part
    return initials + numericPart;
}
```

```
    private static String getInitials(String fullName) {
        StringBuilder initials = new StringBuilder();

        // Split the full name into words
        String[] words = fullName.split("\\s+");

        // Extract the first character of each word as initials
        for (String word : words) {
            if (!word.isEmpty()) {
                initials.append(word.charAt(0));
            }
        }

        return initials.toString();
    }

    private static String getNumericPart(String address) {
        // Extract numeric part from the address
        StringBuilder numericPart = new StringBuilder();
        for (char ch : address.toCharArray()) {
            if (Character.isDigit(ch)) {
                numericPart.append(ch);
            }
        }

        return numericPart.toString();
    }
}
```

b. Count the numbers from 1 to n that have 5 as a digit.

```
import java.util.Scanner;

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

        // Prompt the user for input
        System.out.print("Enter a number (n): ");
        int n = scanner.nextInt();

        // Close the scanner to avoid resource leak
        scanner.close();

        // Count the numbers with digit 5 from 1 to n
        int count = countNumbersWithDigit(n);

        // Display the result
        System.out.println("The count of numbers from 1 to " + n + " with digit 5 is: " + count);
    }

    private static int countNumbersWithDigit(int n) {
        int count = 0;

        // Iterate through numbers from 1 to n
        for (int i = 1; i <= n; i++) {
            // Check if the number contains the digit 5
            if (hasDigit5(i)) {
                count++;
            }
```

```
        }

        return count;
    }

    private static boolean hasDigit5(int number) {
        // Convert the number to a string to check for the digit 5
        String numberString = Integer.toString(number);
        return numberString.contains("5");
    }
}
```

5. Define an interface "QueueOperations" which declares methods for a static queue. Define a class "MyQueue" which contains an array and front and rear as data members and implements the above interface. Initialize the queue using a constructor. Write the code to perform operations on a queue object.

```
// Define the QueueOperations interface
interface QueueOperations {
    void enqueue(int item);    // Add an element to the queue
    int dequeue();             // Remove and return an element from the front of the queue
    void display();            // Display the elements of the queue
}

// Implement the QueueOperations interface in the MyQueue class
class MyQueue implements QueueOperations {
    private static final int MAX_SIZE = 10;
    private int[] queueArray;
    private int front;
    private int rear;

    // Constructor to initialize the queue
```

```
public MyQueue() {
    queueArray = new int[MAX_SIZE];
    front = -1;
    rear = -1;
}

// Implementing the enqueue method from the interface
@Override
public void enqueue(int item) {
    if (rear == MAX_SIZE - 1) {
        System.out.println("Queue is full. Cannot enqueue " + item);
    } else {
        // Increment rear and add the item to the queue
        queueArray[++rear] = item;
        if (front == -1) {
            front = 0;
        }
        System.out.println(item + " enqueued to the queue.");
    }
}

// Implementing the dequeue method from the interface
@Override
public int dequeue() {
    if (front == -1) {
        System.out.println("Queue is empty. Cannot dequeue.");
        return -1;
    } else {
        int dequeuedItem = queueArray[front++];
```

```
            if (front > rear) {
                // Reset front and rear when the queue becomes empty
                front = rear = -1;
            }
            System.out.println(dequeuedItem + " dequeued from the queue.");
            return dequeuedItem;
        }
    }

    // Implementing the display method from the interface
    @Override
    public void display() {
        if (front == -1) {
            System.out.println("Queue is empty.");
        } else {
            System.out.print("Queue elements: ");
            for (int i = front; i <= rear; i++) {
                System.out.print(queueArray[i] + " ");
            }
            System.out.println();
        }
    }
}

// Main class to test the MyQueue class
public class QueueExample {
    public static void main(String[] args) {
        MyQueue myQueue = new MyQueue();
```

```
        myQueue.enqueue(10);
        myQueue.enqueue(20);
        myQueue.enqueue(30);
        myQueue.display();

        myQueue.dequeue();
        myQueue.display();

        myQueue.enqueue(40);
        myQueue.display();
    }
}
```

6. Write a java class called 'student' with name, and rollno. Write a class 'Result' to get Marks of 3 subjects and another class "Sports' to get the points obtained in sports. Calculate the total Marks and displays the result (pass or fail) with points obtained in sports for three students using inheritance and constructor

```
// Define the base class Student
class Student {
    String name;
    int rollNo;

    // Constructor to initialize name and rollNo
    public Student(String name, int rollNo) {
        this.name = name;
        this.rollNo = rollNo;
    }

    // Display method to print student details
    public void displayDetails() {
        System.out.println("Student Name: " + name);
```

```
        System.out.println("Roll Number: " + rollNo);
    }
}

// Define the Result class inheriting from Student
class Result extends Student {
    int marks1, marks2, marks3;

    // Constructor to initialize student details and marks
    public Result(String name, int rollNo, int marks1, int marks2, int marks3) {
        super(name, rollNo);
        this.marks1 = marks1;
        this.marks2 = marks2;
        this.marks3 = marks3;
    }

    // Calculate and display total marks
    public void displayResult() {
        displayDetails();
        int totalMarks = marks1 + marks2 + marks3;
        System.out.println("Total Marks: " + totalMarks);
        System.out.println("Result: " + (totalMarks >= 150 ? "Pass" : "Fail"));
    }
}

// Define the Sports class inheriting from Student
class Sports extends Student {
    int sportsPoints;
```

```
    // Constructor to initialize student details and sports points
    public Sports(String name, int rollNo, int sportsPoints) {
        super(name, rollNo);
        this.sportsPoints = sportsPoints;
    }

    // Display sports points
    public void displaySportsPoints() {
        displayDetails();
        System.out.println("Sports Points: " + sportsPoints);
    }
}

// Main class to test the inheritance and constructors
public class StudentTest {
    public static void main(String[] args) {
        // Create instances of Result and Sports classes
        Result student1Result = new Result("John", 101, 80, 60, 70);
        Sports student1Sports = new Sports("John", 101, 15);

        Result student2Result = new Result("Alice", 102, 90, 85, 80);
        Sports student2Sports = new Sports("Alice", 102, 10);

        Result student3Result = new Result("Bob", 103, 65, 75, 55);
        Sports student3Sports = new Sports("Bob", 103, 20);

        // Display results and sports points for each student
        System.out.println("Student 1 Result:");
        student1Result.displayResult();
```

```
        System.out.println("Sports Points:");
        student1Sports.displaySportsPoints();

        System.out.println("\nStudent 2 Result:");
        student2Result.displayResult();
        System.out.println("Sports Points:");
        student2Sports.displaySportsPoints();

        System.out.println("\nStudent 3 Result:");
        student3Result.displayResult();
        System.out.println("Sports Points:");
        student3Sports.displaySportsPoints();
    }
}
```

7. Define an abstract class "car" with members reg_no, model, reg_date. Define two subclasses of this class – "transportVehicles " (validity_no, start_date, period) and "privateVehicle " (owner_name, owner_address). Define appropriate constructors. Create n objects which could be of either transportVehicles or privateVehicle class by asking the user's choice. Display details of all "privateVehicle" objects and all "transportVehicles" objects.

```
import java.util.Scanner;

// Define the abstract class Car
abstract class Car {
    String regNo;
    String model;
    String regDate;

    // Constructor for Car class
    public Car(String regNo, String model, String regDate) {
        this.regNo = regNo;
```

```
        this.model = model;
        this.regDate = regDate;
    }

    // Abstract method to be implemented by subclasses
    abstract void displayDetails();
}

// Define the subclass TransportVehicle
class TransportVehicle extends Car {
    String validityNo;
    String startDate;
    int period;

    // Constructor for TransportVehicle class
    public TransportVehicle(String regNo, String model, String regDate, String validityNo, String startDate,
int period) {
        super(regNo, model, regDate);
        this.validityNo = validityNo;
        this.startDate = startDate;
        this.period = period;
    }

    // Implementation of displayDetails for TransportVehicle
    @Override
    void displayDetails() {
        System.out.println("Transport Vehicle Details:");
        System.out.println("Registration Number: " + regNo);
        System.out.println("Model: " + model);
```

```
        System.out.println("Registration Date: " + regDate);
        System.out.println("Validity Number: " + validityNo);
        System.out.println("Start Date: " + startDate);
        System.out.println("Period: " + period + " years");
        System.out.println();
    }
}

// Define the subclass PrivateVehicle
class PrivateVehicle extends Car {
    String ownerName;
    String ownerAddress;

    // Constructor for PrivateVehicle class
    public PrivateVehicle(String regNo, String model, String regDate, String ownerName, String
ownerAddress) {
        super(regNo, model, regDate);
        this.ownerName = ownerName;
        this.ownerAddress = ownerAddress;
    }

    // Implementation of displayDetails for PrivateVehicle
    @Override
    void displayDetails() {
        System.out.println("Private Vehicle Details:");
        System.out.println("Registration Number: " + regNo);
        System.out.println("Model: " + model);
        System.out.println("Registration Date: " + regDate);
        System.out.println("Owner Name: " + ownerName);
```

```
        System.out.println("Owner Address: " + ownerAddress);
        System.out.println();
    }
}

// Main class to create objects and display details
public class CarTest {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of vehicles: ");
        int n = scanner.nextInt();

        Car[] vehicles = new Car[n];

        for (int i = 0; i < n; i++) {
            System.out.println("Enter details for vehicle #" + (i + 1));

            System.out.print("Enter registration number: ");
            String regNo = scanner.next();

            System.out.print("Enter model: ");
            String model = scanner.next();

            System.out.print("Enter registration date: ");
            String regDate = scanner.next();

            System.out.println("Choose the type of vehicle:");
            System.out.println("1. Transport Vehicle");
```

```
        System.out.println("2. Private Vehicle");
        System.out.print("Enter your choice (1 or 2): ");
        int choice = scanner.nextInt();

        if (choice == 1) {
            System.out.print("Enter validity number: ");
            String validityNo = scanner.next();

            System.out.print("Enter start date: ");
            String startDate = scanner.next();

            System.out.print("Enter period (in years): ");
            int period = scanner.nextInt();

            vehicles[i] = new TransportVehicle(regNo, model, regDate, validityNo, startDate, period);
        } else if (choice == 2) {
            System.out.print("Enter owner name: ");
            String ownerName = scanner.next();

            System.out.print("Enter owner address: ");
            String ownerAddress = scanner.next();

            vehicles[i] = new PrivateVehicle(regNo, model, regDate, ownerName, ownerAddress);
        } else {
            System.out.println("Invalid choice. Please choose 1 or 2.");
            i--;  // Decrement the loop counter to re-enter details for this iteration
        }
    }
```

```
        // Display details of all Private Vehicles
        System.out.println("Details of Private Vehicles:");
        for (Car vehicle : vehicles) {
            if (vehicle instanceof PrivateVehicle) {
                vehicle.displayDetails();
            }
        }

        // Display details of all Transport Vehicles
        System.out.println("Details of Transport Vehicles:");
        for (Car vehicle : vehicles) {
            if (vehicle instanceof TransportVehicle) {
                vehicle.displayDetails();
            }
        }

        scanner.close();
    }
}
```

8. Create an interface “CreditCardInterface” with methods to viewCreditAmount, viewPin, changePin and payBalance. Create a class Customer (name, card number, pin, creditAmount – initialized to 0). Implement methods of the interface “CreditCardInterface” in Customer class. Create an array of customer objects and perform the following actions.

- **Pay Balance**
- **Change Pin**

```
// Define the CreditCardInterface
interface CreditCardInterface {
    void viewCreditAmount();
    void viewPin();
```

```
    void changePin(int newPin);
    void payBalance(double amount);
}

// Implement the CreditCardInterface in the Customer class
class Customer implements CreditCardInterface {
    String name;
    long cardNumber;
    int pin;
    double creditAmount;

    // Constructor for Customer class
    public Customer(String name, long cardNumber, int pin) {
        this.name = name;
        this.cardNumber = cardNumber;
        this.pin = pin;
        this.creditAmount = 0.0;
    }

    // Implementation of viewCreditAmount
    @Override
    public void viewCreditAmount() {
        System.out.println("Credit Amount for " + name + ": $" + creditAmount);
    }

    // Implementation of viewPin
    @Override
    public void viewPin() {
        System.out.println("PIN for " + name + ": " + pin);
```

```
    }

    // Implementation of changePin
    @Override
    public void changePin(int newPin) {
        pin = newPin;
        System.out.println("PIN changed successfully for " + name);
    }

    // Implementation of payBalance
    @Override
    public void payBalance(double amount) {
        if (amount > 0 && amount <= creditAmount) {
            creditAmount -= amount;
            System.out.println("Payment of $" + amount + " made successfully for " + name);
        } else {
            System.out.println("Invalid payment amount or insufficient credit for " + name);
        }
    }
}

// Main class to perform actions on an array of customer objects
public class CreditCardTest {
    public static void main(String[] args) {
        // Create an array of customer objects
        Customer[] customers = new Customer[3];
        customers[0] = new Customer("John", 1234567890123456L, 1234);
        customers[1] = new Customer("Alice", 9876543210987654L, 5678);
        customers[2] = new Customer("Bob", 1111222233334444L, 4321);
```

```
        // Perform actions on customer objects
        for (Customer customer : customers) {
            System.out.println("\nActions for customer: " + customer.name);
            customer.viewCreditAmount();
            customer.viewPin();
            customer.changePin(5678);
            customer.viewPin();
            customer.payBalance(50.0);
            customer.viewCreditAmount();
        }
    }
}
```

9. Write a Java program to perform the following task. • Take an integer array of size 20, initialize values randomly between 10 and 90, simultaneously sum all values and calculate average. Now separate values below average and above average in ArrayLists. Finally print both lists in 2 separate rows.

```
import java.util.ArrayList;
import java.util.Random;

public class ArraySeparation {
    public static void main(String[] args) {
        // Initialize an integer array of size 20 with random values between 10 and 90
        int[] numbers = new int[20];
        Random random = new Random();

        for (int i = 0; i < numbers.length; i++) {
            numbers[i] = random.nextInt(81) + 10;  // Generates random number between 10 and 90
        }
```

```
// Calculate the sum and average of the array
int sum = 0;
for (int number : numbers) {
    sum += number;
}
double average = (double) sum / numbers.length;

// Separate values below and above average into ArrayLists
ArrayList<Integer> belowAverageList = new ArrayList<>();
ArrayList<Integer> aboveAverageList = new ArrayList<>();

for (int number : numbers) {
    if (number < average) {
        belowAverageList.add(number);
    } else {
        aboveAverageList.add(number);
    }
}

// Print the original array
System.out.println("Original Array:");
printArray(numbers);

// Print the below average values
System.out.println("\nValues Below Average:");
printArrayList(belowAverageList);

// Print the above average values
System.out.println("\nValues Above Average:");
```

```
        printArrayList(aboveAverageList);
    }

    // Utility method to print an integer array
    private static void printArray(int[] array) {
        for (int num : array) {
            System.out.print(num + " ");
        }
        System.out.println();
    }

    // Utility method to print an ArrayList
    private static void printArrayList(ArrayList<Integer> list) {
        for (int num : list) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

10. Write a java program that reads a string from inputs containing first name, last name and computes an e-mail address with first 3 letters of the first name, first 4 letters of last name, '.' separator and domain. Display the outputs by invoking objects. Create a java abstract class to implement stack concept. Check for the overflow and empty conditions.

```
import java.util.Scanner;

// Define the abstract class Stack
abstract class Stack {
    protected int maxSize;
    protected int top;
    protected String[] stackArray;
```

```
    // Constructor to initialize the stack
    public Stack(int size) {
        maxSize = size;
        top = -1;
        stackArray = new String[maxSize];
    }

    // Abstract methods to be implemented by subclasses
    public abstract void push(String item);
    public abstract String pop();
    public abstract boolean isEmpty();
    public abstract boolean isFull();
}

// Implement the Stack abstract class
class StringStack extends Stack {
    // Constructor for StringStack
    public StringStack(int size) {
        super(size);
    }

    // Implementation of push method
    @Override
    public void push(String item) {
        if (!isFull()) {
            stackArray[++top] = item;
            System.out.println("Pushed: " + item);
        } else {
```

```
            System.out.println("Stack Overflow! Cannot push " + item);
        }
    }

    // Implementation of pop method
    @Override
    public String pop() {
        if (!isEmpty()) {
            System.out.println("Popped: " + stackArray[top]);
            return stackArray[top--];
        } else {
            System.out.println("Stack Underflow! Cannot pop.");
            return null;
        }
    }

    // Implementation of isEmpty method
    @Override
    public boolean isEmpty() {
        return top == -1;
    }

    // Implementation of isFull method
    @Override
    public boolean isFull() {
        return top == maxSize - 1;
    }
}
```

```
// Main class to read names, compute email address, and demonstrate stack operations
public class EmailAddressAndStack {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Read first name and last name
        System.out.print("Enter your first name: ");
        String firstName = scanner.nextLine();

        System.out.print("Enter your last name: ");
        String lastName = scanner.nextLine();

        // Compute email address
        String emailAddress = computeEmailAddress(firstName, lastName);
        System.out.println("Computed Email Address: " + emailAddress);

        // Demonstrate stack operations
        StringStack stack = new StringStack(5);

        // Push names onto the stack
        stack.push(firstName);
        stack.push(lastName);

        // Pop names from the stack
        stack.pop();
        stack.pop();

        scanner.close();
    }
```

```
    // Method to compute email address based on the specified format
    private static String computeEmailAddress(String firstName, String lastName) {
        String firstPart = firstName.substring(0, Math.min(3, firstName.length())).toLowerCase();
        String secondPart = lastName.substring(0, Math.min(4, lastName.length())).toLowerCase();
        String domain = "example.com"; // Change this to the desired domain

        return firstPart + "." + secondPart + "@" + domain;
    }
}
```

11. Write a java program for exception handling:

a. To create a user defined exception whenever user input the word “hello”.

```
import java.util.Scanner;

// Define a custom exception class
class HelloInputException extends Exception {
    public HelloInputException(String message) {
        super(message);
    }
}

// Main class to demonstrate exception handling
public class ExceptionHandlingExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            // Prompt the user for input
            System.out.print("Enter a word: ");
```

```
            String userInput = scanner.next();

            // Check if the input is "hello" and throw the custom exception if true
            if ("hello".equalsIgnoreCase(userInput)) {
                throw new HelloInputException("Input cannot be 'hello'.");
            }

            // If the input is not "hello", display a message
            System.out.println("User input: " + userInput);

        } catch (HelloInputException e) {
            // Handle the custom exception
            System.out.println("Exception: " + e.getMessage());

        } finally {
            // Close the scanner to avoid resource leak
            scanner.close();
        }
    }
}
```

b. To add two integers and raise exception when any other character except number (0 – 9) is given as input.

```
import java.util.Scanner;

// Custom exception class for handling non-numeric input
class NonNumericInputException extends Exception {
    public NonNumericInputException(String message) {
        super(message);
```

```
    }
}

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

        try {
            // Prompt the user for the first integer
            System.out.print("Enter the first integer: ");
            int num1 = readInteger(scanner);

            // Prompt the user for the second integer
            System.out.print("Enter the second integer: ");
            int num2 = readInteger(scanner);

            // Add the two integers
            int result = num1 + num2;

            // Display the result
            System.out.println("Sum: " + result);

        } catch (NonNumericInputException e) {
            // Handle the custom exception for non-numeric input
            System.out.println("Exception: " + e.getMessage());

        } finally {
            // Close the scanner to avoid resource leak
            scanner.close();
```

```
    }
  }

  // Method to read an integer from the user, handling non-numeric input
  private static int readInteger(Scanner scanner) throws NonNumericInputException {
    try {
      // Read the input as a string
      String input = scanner.next();

      // Try parsing the input as an integer
      return Integer.parseInt(input);

    } catch (NumberFormatException e) {
      // If parsing fails, throw the custom exception for non-numeric input
      throw new NonNumericInputException("Non-numeric input detected. Please enter a valid integer.");
    }
  }
}
```

12. Create a class Doctor with attributes id, name, age and department. Initialize values through parameterized constructor. If age of Doctor is not in between 25 and 65 then generate user-defined exception "AgeNotWithinRangeException". If name contains numbers or special symbols raise exception "NameNotValidException". Define the two exception classes.

```
// Custom exception class for age not within range
class AgeNotWithinRangeException extends Exception {
  public AgeNotWithinRangeException(String message) {
    super(message);
  }
}
```

```
// Custom exception class for name not valid
class NameNotValidException extends Exception {
    public NameNotValidException(String message) {
        super(message);
    }
}

// Class representing a Doctor
class Doctor {
    private int id;
    private String name;
    private int age;
    private String department;

    // Parameterized constructor to initialize Doctor attributes
    public Doctor(int id, String name, int age, String department) throws AgeNotWithinRangeException,
NameNotValidException {
        // Validate age
        if (age < 25 || age > 65) {
            throw new AgeNotWithinRangeException("Age should be between 25 and 65.");
        }

        // Validate name
        if (!isValidName(name)) {
            throw new NameNotValidException("Name should not contain numbers or special symbols.");
        }

        // Initialize attributes
        this.id = id;
```

```
        this.name = name;
        this.age = age;
        this.department = department;
    }

    // Method to check if the name is valid (does not contain numbers or special symbols)
    private boolean isValidName(String name) {
        return name.matches("[a-zA-Z]+");
    }

    // Getter methods
    public int getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public String getDepartment() {
        return department;
    }
}

// Main class to demonstrate Doctor class and exception handling
```

```
public class DoctorTest {
    public static void main(String[] args) {
        try {
            // Creating a Doctor object with valid attributes
            Doctor doctor1 = new Doctor(1, "John Doe", 35, "Cardiology");
            System.out.println("Doctor 1 details:");
            displayDoctorDetails(doctor1);

            // Creating a Doctor object with age not within range
            // This should throw AgeNotWithinRangeException
            Doctor doctor2 = new Doctor(2, "Jane Smith", 70, "Orthopedics");  // This line should throw an exception

        } catch (AgeNotWithinRangeException | NameNotValidException e) {
            // Handle the custom exceptions
            System.out.println("Exception: " + e.getMessage());
        }
    }

    // Utility method to display Doctor details
    private static void displayDoctorDetails(Doctor doctor) {
        System.out.println("Doctor ID: " + doctor.getId());
        System.out.println("Doctor Name: " + doctor.getName());
        System.out.println("Doctor Age: " + doctor.getAge());
        System.out.println("Doctor Department: " + doctor.getDepartment());
        System.out.println();
    }
}
```

13. A program accepts two integers as command line arguments. It displays all prime numbers between these two. Validate the input for the following criteria: Both should be positive integers. The second should be larger than the first. Create user defined exceptions for both.

```
// Custom exception class for negative numbers
class NegativeNumberException extends Exception {
    public NegativeNumberException(String message) {
        super(message);
    }
}

// Custom exception class for invalid range
class InvalidRangeException extends Exception {
    public InvalidRangeException(String message) {
        super(message);
    }
}

// Main class to find prime numbers in the given range
public class PrimeNumberRange {
    public static void main(String[] args) {
        try {
            // Validate command line arguments
            if (args.length != 2) {
                throw new IllegalArgumentException("Please provide exactly two integers as command line arguments.");
            }

            int num1 = validatePositiveInteger(args[0]);
            int num2 = validatePositiveInteger(args[1]);
```

```
            validateNumberRange(num1, num2);

            // Display prime numbers in the given range
            System.out.println("Prime numbers between " + num1 + " and " + num2 + ":");
            displayPrimeNumbers(num1, num2);

        } catch (NumberFormatException | IllegalArgumentException e) {
            System.out.println("Exception: " + e.getMessage());
        } catch (NegativeNumberException | InvalidRangeException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }

    // Method to validate a positive integer
    private static int validatePositiveInteger(String arg) throws NegativeNumberException {
        int num = Integer.parseInt(arg);

        if (num < 0) {
            throw new NegativeNumberException("Negative number not allowed: " + num);
        }

        return num;
    }

    // Method to validate the number range
    private static void validateNumberRange(int num1, int num2) throws InvalidRangeException {
        if (num2 <= num1) {
            throw new InvalidRangeException("Second number should be larger than the first.");
        }
```

```
    }

    // Method to display prime numbers in the given range
    private static void displayPrimeNumbers(int start, int end) {
        for (int i = start; i <= end; i++) {
            if (isPrime(i)) {
                System.out.print(i + " ");
            }
        }
        System.out.println();
    }

    // Method to check if a number is prime
    private static boolean isPrime(int num) {
        if (num <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

14. Write a Java program 'WordCount' that counts the words in one or more files. Start a new thread for each file. For example, if you call

"java WordCount report.txt address.txt Homework.java " then the program might print

address.txt: 1052

Homework.java: 445

report.txt: 2099

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

class WordCounter extends Thread {
    private String fileName;

    public WordCounter(String fileName) {
        this.fileName = fileName;
    }

    @Override
    public void run() {
        try {
            int wordCount = countWords(fileName);
            System.out.println(fileName + ": " + wordCount + " words");

        } catch (IOException e) {
            System.out.println("Error reading file: " + fileName);
        }
    }

    private int countWords(String fileName) throws IOException {
        int wordCount = 0;

        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
```

```
            String line;
            while ((line = reader.readLine()) != null) {
                String[] words = line.split("\\s+"); // Split by whitespace
                wordCount += words.length;
            }
        }

        return wordCount;
    }
}

public class WordCount {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Usage: java WordCount <file1> <file2> ...");
            return;
        }

        Thread[] threads = new Thread[args.length];

        for (int i = 0; i < args.length; i++) {
            threads[i] = new WordCounter(args[i]);
            threads[i].start();
        }

        // Wait for all threads to finish
        for (Thread thread : threads) {
            try {
                thread.join();
```

```
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted: " + e.getMessage());
            }
        }
    }
}
```

15. Write a Java program 'LineCounts.java' that will count the number of lines in each files that is specified on the command line. Note that multiple files can be specified, as in

"java LineCounts file1.txt file2.txt file3.txt".

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class LineCounts {
    public static void main(String[] args) {
        if (args.length == 0) {
            System.out.println("Usage: java LineCounts <file1> <file2> ...");
            return;
        }

        for (String fileName : args) {
            int lineCount = countLines(fileName);
            System.out.println(fileName + ": " + lineCount + " lines");
        }
    }

    private static int countLines(String fileName) {
        int lineCount = 0;
```

```
        try (BufferedReader reader = new BufferedReader(new FileReader(fileName))) {
            while (reader.readLine() != null) {
                lineCount++;
            }
        } catch (IOException e) {
            System.out.println("Error reading file: " + fileName);
        }

        return lineCount;
    }
}
```

16.Write a java program to find the minimum, maximum value from the given type of elements using a generic function.

```
public class MinMaxFinder<T extends Comparable<T>> {

    public MinMaxResult<T> findMinMax(T[] array) {
        if (array == null || array.length == 0) {
            return null;
        }

        T min = array[0];
        T max = array[0];

        for (int i = 1; i < array.length; i++) {
            if (array[i].compareTo(min) < 0) {
                min = array[i];
            }
            if (array[i].compareTo(max) > 0) {
                max = array[i];
```

```
            }
        }

        return new MinMaxResult<>(min, max);
    }

    public static void main(String[] args) {
        Integer[] intArray = {5, 2, 8, 1, 7, 3};
        Double[] doubleArray = {3.5, 1.2, 7.8, 2.3, 5.4};
        String[] stringArray = {"apple", "orange", "banana", "grape"};

        MinMaxFinder<Integer> intFinder = new MinMaxFinder<>();
        MinMaxFinder<Double> doubleFinder = new MinMaxFinder<>();
        MinMaxFinder<String> stringFinder = new MinMaxFinder<>();

        MinMaxResult<Integer> intResult = intFinder.findMinMax(intArray);
        MinMaxResult<Double> doubleResult = doubleFinder.findMinMax(doubleArray);
        MinMaxResult<String> stringResult = stringFinder.findMinMax(stringArray);

        System.out.println("Integer Array - Min: " + intResult.getMin() + ", Max: " + intResult.getMax());
        System.out.println("Double Array - Min: " + doubleResult.getMin() + ", Max: " +
doubleResult.getMax());
        System.out.println("String Array - Min: " + stringResult.getMin() + ", Max: " + stringResult.getMax());
    }
}

// Class to store the result of finding min and max
class MinMaxResult<T> {
    private T min;
```

```
    private T max;

    public MinMaxResult(T min, T max) {
        this.min = min;
        this.max = max;
    }

    public T getMin() {
        return min;
    }

    public T getMax() {
        return max;
    }
}
```

17. a. Write a Java program to demonstrate that as a high-priority thread executes, it will delay the execution of all lower-priority threads.

```
public class PriorityExample {

    public static void main(String[] args) {
        // Create a high-priority thread
        Thread highPriorityThread = new Thread(new PriorityTask("HighPriorityTask"));
        highPriorityThread.setPriority(Thread.MAX_PRIORITY);

        // Create two low-priority threads
        Thread lowPriorityThread1 = new Thread(new PriorityTask("LowPriorityTask1"));
        Thread lowPriorityThread2 = new Thread(new PriorityTask("LowPriorityTask2"));
        lowPriorityThread1.setPriority(Thread.MIN_PRIORITY);
        lowPriorityThread2.setPriority(Thread.MIN_PRIORITY);
```

```
    // Start the threads
    highPriorityThread.start();
    lowPriorityThread1.start();
    lowPriorityThread2.start();
}

static class PriorityTask implements Runnable {
    private String taskName;

    public PriorityTask(String taskName) {
        this.taskName = taskName;
    }

    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("Task: " + taskName + ", Iteration: " + i +
                    ", Priority: " + Thread.currentThread().getPriority());

            // Simulate some work
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

}

b. Write a Java program to read from an input file and convert the words to lower case and write it in another file.

```
import java.io.BufferedReader;

import java.io.BufferedWriter;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;


public class ConvertToLowerCase {


   public static void main(String[] args) {

      if (args.length != 2) {

         System.out.println("Usage: java ConvertToLowerCase <inputFileName> <outputFileName>");

         return;

      }


      String inputFileName = args[0];

      String outputFileName = args[1];


      try {

         // Read from the input file

         BufferedReader reader = new BufferedReader(new FileReader(inputFileName));


         // Write to the output file

         BufferedWriter writer = new BufferedWriter(new FileWriter(outputFileName));


         String line;

         while ((line = reader.readLine()) != null) {
```

```
            String[] words = line.split("\\s+"); // Split by whitespace
            for (String word : words) {
                // Convert each word to lowercase and write to the output file
                writer.write(word.toLowerCase() + " ");
            }
            writer.newLine(); // Move to the next line
        }

        // Close the reader and writer
        reader.close();
        writer.close();

        System.out.println("Conversion completed successfully.");

    } catch (IOException e) {
        System.out.println("Error: " + e.getMessage());
    }
  }
}
```

18.Write java programs that include generic method to satisfy the following property.

a. To counts the number of odd integers in an integer list

```
import java.util.List;

public class GenericMethods {

  public static <T extends Number> int countOddIntegers(List<T> list) {
    int count = 0;
    for (T element : list) {
      if (element.intValue() % 2 != 0) {
```

```
            count++;
        }
    }
    return count;
}

public static void main(String[] args) {
    // Example usage for counting odd integers in a list
    List<Integer> integerList = List.of(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
    int oddCount = countOddIntegers(integerList);
    System.out.println("Number of odd integers: " + oddCount);
}
}
```

b. To exchange the positions of two different elements in an array.

```
public class GenericMethods {

    public static <T> void exchangeElements(T[] array, int index1, int index2) {
        if (index1 >= 0 && index1 < array.length && index2 >= 0 && index2 < array.length && index1 != index2) {
            T temp = array[index1];
            array[index1] = array[index2];
            array[index2] = temp;
        }
    }

    public static void main(String[] args) {
        // Example usage for exchanging positions in an array
        Integer[] intArray = {1, 2, 3, 4, 5};
```

```
        System.out.println("Original Array: " + java.util.Arrays.toString(intArray));
        exchangeElements(intArray, 1, 3);
        System.out.println("Array after exchange: " + java.util.Arrays.toString(intArray));
    }
}
```

c. To find the maximal element in the range [begin, end] of a list.

```
import java.util.List;

public class GenericMethods {

    public static <T extends Comparable<T>> T findMaxElementInRange(List<T> list, int begin, int end) {
        if (list == null || begin < 0 || end >= list.size() || begin > end) {
            return null;
        }

        T maxElement = list.get(begin);
        for (int i = begin + 1; i <= end; i++) {
            if (list.get(i).compareTo(maxElement) > 0) {
                maxElement = list.get(i);
            }
        }

        return maxElement;
    }

    public static void main(String[] args) {
        // Example usage for finding the maximal element in a range of a list
        List<Integer> integerList = List.of(3, 8, 5, 12, 7, 10, 6);
        int begin = 1;
```

```
        int end = 4;
        Integer maxElementInRange = findMaxElementInRange(integerList, begin, end);
        System.out.println("Maximal element in the range [" + begin + ", " + end + "]: " +
maxElementInRange);
    }
}
```

19. Create a new Java GUI application to convert miles to kilometers when pressing the "Convert!" button. Note that you need to implement the ActionListener interface and override the actionPerformed() method. Note that 1 mile is equal to 1.609 kilometers.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class MilesToKilometersConverter extends JFrame implements ActionListener {

    private JTextField milesTextField;
    private JLabel resultLabel;

    public MilesToKilometersConverter() {
        // Set up the JFrame
        setTitle("Miles to Kilometers Converter");
        setSize(300, 150);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Create components
        milesTextField = new JTextField(10);
        resultLabel = new JLabel("Kilometers: ");
```

```
    JButton convertButton = new JButton("Convert!");
    convertButton.addActionListener(this);

    // Set up layout
    setLayout(new GridLayout(3, 1));
    add(milesTextField);
    add(convertButton);
    add(resultLabel);
}

@Override
public void actionPerformed(ActionEvent e) {
    // Handle button click event
    if (e.getActionCommand().equals("Convert!")) {
        convertMilesToKilometers();
    }
}

private void convertMilesToKilometers() {
    try {
        // Get miles from user input
        double miles = Double.parseDouble(milesTextField.getText());

        // Convert miles to kilometers
        double kilometers = miles * 1.609;

        // Display the result
        resultLabel.setText("Kilometers: " + kilometers);
```

```
        } catch (NumberFormatException ex) {
            // Handle non-numeric input
            resultLabel.setText("Invalid input. Please enter a number.");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            MilesToKilometersConverter converter = new MilesToKilometersConverter();
            converter.setVisible(true);
        });
    }
}
```

20. Develop a course registration form with Name, Address, phone number,Gender(Male or Female),department(user have to select from CSE, ECE,EEE, Mech, Civil) and course (user have to select from (C,C++,JAVA,PYTHON). When the user submits the form, a dialog box should appear with a message "Username , you have successfully enrolled inCourse Name

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class CourseRegistrationForm extends JFrame implements ActionListener {

    private JTextField nameField, addressField, phoneField;
    private JComboBox<String> genderComboBox, departmentComboBox, courseComboBox;

    public CourseRegistrationForm() {
        // Set up the JFrame
        setTitle("Course Registration Form");
```

```
setSize(400, 300);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);

// Create components
nameField = new JTextField(20);
addressField = new JTextField(20);
phoneField = new JTextField(20);

String[] genders = {"Male", "Female"};
genderComboBox = new JComboBox<>(genders);

String[] departments = {"CSE", "ECE", "EEE", "Mech", "Civil"};
departmentComboBox = new JComboBox<>(departments);

String[] courses = {"C", "C++", "JAVA", "PYTHON"};
courseComboBox = new JComboBox<>(courses);

JButton submitButton = new JButton("Submit");
submitButton.addActionListener(this);

// Set up layout
setLayout(new GridLayout(7, 2));
add(new JLabel("Name:"));
add(nameField);
add(new JLabel("Address:"));
add(addressField);
add(new JLabel("Phone Number:"));
add(phoneField);
```

```
        add(new JLabel("Gender:"));
        add(genderComboBox);
        add(new JLabel("Department:"));
        add(departmentComboBox);
        add(new JLabel("Course:"));
        add(courseComboBox);
        add(submitButton);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("Submit")) {
            displayConfirmationDialog();
        }
    }

    private void displayConfirmationDialog() {
        String name = nameField.getText();
        String department = (String) departmentComboBox.getSelectedItem();
        String course = (String) courseComboBox.getSelectedItem();

        String message = String.format("%s, you have successfully enrolled in %s course.", name, course);

        JOptionPane.showMessageDialog(this, message, "Registration Confirmation",
JOptionPane.INFORMATION_MESSAGE);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
```

```
            CourseRegistrationForm registrationForm = new CourseRegistrationForm();
            registrationForm.setVisible(true);
        });
    }
}
```