OOP FISAC

1

import java.util.\*;

class SortUtility<T extends Comparable<T>> {

    private T[] data;

    public SortUtility(T[] data) {

        this.data = data;

    }

    public void sort() {

        Arrays.sort(data);

    }

    public void display() {

        for (T item : data) {

            System.out.println(item.toString());

        }

    }

}

class Employee implements Comparable<Employee> {

    private String name;

    private int YoE;

    public Employee(String name, int YoE) {

        this.name = name;

        this.YoE = YoE;

    }

    @Override

    public int compareTo(Employee other) {

        return this.YoE - other.YoE;

    }

    @Override

    public String toString() {

        return "Employee [Name: " + name + ", YoE: " + YoE + "]";

    }

}

class Student implements Comparable<Student> {

    private String name;

    private String grade;

    public Student(String name, String grade) {

        this.name = name;

        this.grade = grade;

    }

    @Override

    public int compareTo(Student other) {

        return this.grade.compareTo(other.grade);

    }

    @Override

    public String toString() {

        return "Student [Name: " + name + ", Grade: " + grade + "]";

    }

}

 class Main {

    public static void main(String[] args) {

        Employee[] employees = {

            new Employee("Avani", 5),

            new Employee("Naman", 3),

            new Employee("Jay", 8)

        };

        Student[] students = {

            new Student("Shambhavi", "B"),

            new Student("Ansh", "A"),

            new Student("Lian", "C")

        };

        SortUtility<Employee> employeeSorter = new SortUtility<>(employees);

        employeeSorter.sort();

        System.out.println("Sorted Employees:");

        employeeSorter.display();

        SortUtility<Student> studentSorter = new SortUtility<>(students);

        studentSorter.sort();

        System.out.println("Sorted Students:");

        studentSorter.display();

    }

}

Output

Sorted Employees:

Employee [Name: Naman, YoE: 3]

Employee [Name: Avani, YoE: 5]

Employee [Name: Jay, YoE: 8]

Sorted Students:

Student [Name: Ansh, Grade: A]

Student [Name: Shambhavi, Grade: B]

Student [Name: Lian, Grade: C]

2

class VowelException extends Exception {

    public VowelException(String message) {

        super(message);

    }

}

class WordAnalysis {

    public static void main(String[] args) {

        String sentence = "The quick brown fox jumps  the lazy dog";

        Thread countThread = new Thread(() -> {

            int vowelWordCount = 0;

            String[] words = sentence.split(" ");

            for (String word : words) {

                if (word.matches("[aeiouAEIOU].\*")) {

                    vowelWordCount++;

                }

            }

            if (vowelWordCount == 0) {

                try {

                    throw new VowelException("Word not found which starts with a vowel.");

                } catch (VowelException e) {

                    System.err.println(e.getMessage());

                }

            } else {

                System.out.println("Number of words starting with vowels: " + vowelWordCount);

            }

        });

        Thread lengthThread = new Thread(() -> {

            String[] words = sentence.split(" ");

            for (String word : words) {

                System.out.println("Word: " + word + ", Length: " + word.length());

            }

        });

        countThread.start();

        lengthThread.start();

        try {

            countThread.join();

            lengthThread.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

    }

}

Output

Number of words starting with vowels: 1

Word: The, Length: 3

Word: quick, Length: 5

Word: brown, Length: 5

Word: fox, Length: 3

Word: jumps, Length: 5

Word: over, Length: 4

Word: the, Length: 3

Word: lazy, Length: 4

Word: dog, Length: 3

3

import java.util.Scanner;

class NegativeMarkException extends Exception {

    public NegativeMarkException(String message) {

        super(message);

    }

}

class OutOfRangeException extends Exception {

    public OutOfRangeException(String message) {

        super(message);

    }

}

class Student {

    String studentName;

    String regNo;

    String category;

    int[] marks = new int[3];

    char grade;

    public Student(String studentName, String regNo, String category) {

        this.studentName = studentName;

        this.regNo = regNo;

        this.category = category;

    }

    public void input() throws NegativeMarkException, OutOfRangeException {

        Scanner scanner = new Scanner(System.in);

        for (int i = 0; i < 3; i++) {

            System.out.print("Enter mark for Subject " + (i + 1) + ": ");

            int mark = scanner.nextInt();

            if (mark < 0) {

                throw new NegativeMarkException("Negative marks are not allowed.");

            } else if (mark < 0 || mark > 100) {

                throw new OutOfRangeException("Marks should be between 0 and 100.");

            }

            marks[i] = mark;

        }

        calculateGrade();

    }

    private void calculateGrade() {

        int total = marks[0] + marks[1] + marks[2];

        int average = total / 3;

        if (average >= 90) {

            grade = 'A';

        } else if (average >= 80) {

            grade = 'B';

        } else if (average >= 70) {

            grade = 'C';

        } else {

            grade = 'F';

        }

    }

    public String getRegNo() {

        return regNo;

    }

    public char getGrade() {

        return grade;

    }

}

 class ResultProcessing {

    public static void main(String[] args) {

        Student[] students = new Student[3];

        try {

            Scanner scanner = new Scanner(System.in);

            String[][] studentDetails = new String[3][2]; // 2D matrix to store RegNo and grade

            for (int i = 0; i < students.length; i++) {

                System.out.println("Enter details for Student " + (i + 1) + ":");

                System.out.print("Student Name: ");

                String studentName = scanner.next();

                System.out.print("Reg No: ");

                String regNo = scanner.next();

                System.out.print("Category (UG/PG): ");

                String category = scanner.next();

                students[i] = new Student(studentName, regNo, category);

                students[i].input();

                // Store RegNo and grade in the 2D matrix

                studentDetails[i][0] = students[i].getRegNo();

                studentDetails[i][1] = String.valueOf(students[i].getGrade());

            }

            // Sort students based on average marks

            for (int i = 0; i < students.length; i++) {

                for (int j = i + 1; j < students.length; j++) {

                    if (students[i].getGrade() > students[j].getGrade()) {

                        Student temp = students[i];

                        students[i] = students[j];

                        students[j] = temp;

                    }

                }

            }

            System.out.println("Top 3 Students with Highest Grades:");

            for (int i = 0; i < Math.min(3, students.length); i++) {

                System.out.println("Student Name: " + students[i].studentName);

                System.out.println("Reg No: " + students[i].getRegNo());

                System.out.println("Category: " + students[i].category);

                System.out.println("Grade: " + students[i].getGrade());

                System.out.println();

            }

            // Display the 2D matrix with RegNo and grade

            System.out.println("2D Matrix of RegNo and Grade:");

            for (int i = 0; i < students.length; i++) {

                System.out.println("Reg No: " + studentDetails[i][0]);

                System.out.println("Grade: " + studentDetails[i][1]);

                System.out.println();

            }

        } catch (NegativeMarkException | OutOfRangeException e) {

            System.err.println("Error: " + e.getMessage());

        }

    }

}

Enter details for Student 1:

Student Name: arjun

Reg No: 5678

Category (UG/PG): UG

Enter mark for Subject 1: 10

Enter mark for Subject 2: 20

Enter mark for Subject 3: 1

Enter details for Student 2:

Student Name: avani

Reg No: 678

Category (UG/PG): UG

Enter mark for Subject 1: 90

Enter mark for Subject 2: 100

Enter mark for Subject 3: 100

Enter details for Student 3:

Student Name: fghj

Reg No: 200

Category (UG/PG): UG

Enter mark for Subject 1: 23

Enter mark for Subject 2: 34

Enter mark for Subject 3: 56

Top 3 Students with Highest Grades:

Student Name: avani

Reg No: 678

Category: UG

Grade: A

Student Name: arjun

Reg No: 5678

Category: UG

Grade: F

Student Name: fghj

Reg No: 200

Category: UG

Grade: F

2D Matrix of RegNo and Grade:

Reg No: 5678

Grade: F

Reg No: 678

Grade: A

Reg No: 200

Grade: F

4

import java.util.Arrays;

import java.util.Scanner;

 class StringCheckUtility {

    private static String[] strings = new String[5];

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        Thread s1 = new Thread(() -> {

            System.out.println("S1 Thread: Enter 5 strings.");

            for (int i = 0; i < 5; i++) {

                strings[i] = scanner.nextLine();

            }

        });

        Thread s2 = new Thread(() -> {

            System.out.println("S2 Thread: Number of vowels and characters in each string:");

            for (String str : strings) {

                int vowels = 0;

                int characters = str.length();

                for (char ch : str.toCharArray()) {

                    if ("aeiouAEIOU".contains(String.valueOf(ch))) {

                        vowels++;

                    }

                }

                System.out.println("String: " + str + " - Vowels: " + vowels + ", Characters: " + characters);

            }

        });

        Thread s3 = new Thread(() -> {

            System.out.println("S3 Thread: Sorting strings in ascending order.");

            Arrays.sort(strings);

        });

        Thread s4 = new Thread(() -> {

            System.out.println("S4 Thread: Enter a substring to search in a string.");

            String substring = scanner.nextLine();

            for (String str : strings) {

                if (str.contains(substring)) {

                    System.out.println("Substring '" + substring + "' found in: " + str);

                }

            }

        });

        s1.start();

        try {

            s1.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        s2.start();

        s3.start();

        s4.start();

        try {

            s2.join();

            s3.join();

            s4.join();

        } catch (InterruptedException e) {

            e.printStackTrace();

        }

        System.out.println("Main Thread: Common vowels present in any two strings:");

        for (int i = 0; i < 4; i++) {

            for (int j = i + 1; j < 5; j++) {

                String str1 = strings[i];

                String str2 = strings[j];

                String commonVowels = getCommonVowels(str1, str2);

                if (!commonVowels.isEmpty()) {

                    System.out.println("Strings: " + str1 + " and " + str2 + " - Common Vowels: " + commonVowels);

                }

            }

        }

    }

    private static String getCommonVowels(String str1, String str2) {

        String commonVowels = "";

        for (char ch : "aeiouAEIOU".toCharArray()) {

            if (str1.contains(String.valueOf(ch)) && str2.contains(String.valueOf(ch))) {

                commonVowels += ch;

            }

        }

        return commonVowels;

    }

}

Output

S1 Thread: Enter 5 strings.

avani

hello

wow

hi

woah

S2 Thread: Number of vowels and characters in each string:

S3 Thread: Sorting strings in ascending order.

S4 Thread: Enter a substring to search in a string.

String: avani - Vowels: 3, Characters: 5

String: hello - Vowels: 2, Characters: 5

String: hi - Vowels: 1, Characters: 2

String: woah - Vowels: 2, Characters: 4

String: wow - Vowels: 1, Characters: 3

5

import java.util.HashSet;

import java.util.Scanner;

import java.util.StringTokenizer;

import java.util.concurrent.atomic.AtomicBoolean;

class NoRepeatingWordsException extends Exception {

    public NoRepeatingWordsException(String message) {

        super(message);

    }

}

 class SentenceProcessing {

    private static final Object lock = new Object();

    private static AtomicBoolean readDone = new AtomicBoolean(false);

    private static AtomicBoolean findRepeatingWordsDone = new AtomicBoolean(false);

    private static AtomicBoolean removeRepeatingWordsDone = new AtomicBoolean(false);

    private static String sentence;

    private static HashSet<String> repeatingWords = new HashSet<>();

    public static void main(String[] args) {

        Thread readerThread = new Thread(() -> {

            synchronized (lock) {

                Scanner scanner = new Scanner(System.in);

                System.out.print("Enter a sentence: ");

                sentence = scanner.nextLine();

                readDone.set(true);

                lock.notifyAll();

            }

        });

        Thread findRepeatingWordsThread = new Thread(() -> {

            synchronized (lock) {

                while (!readDone.get()) {

                    try {

                        lock.wait();

                    } catch (InterruptedException e) {

                        e.printStackTrace();

                    }

                }

                StringTokenizer tokenizer = new StringTokenizer(sentence);

                while (tokenizer.hasMoreTokens()) {

                    String word = tokenizer.nextToken().toLowerCase();

                    if (repeatingWords.contains(word)) {

                        System.out.println("Repeating Word Found: " + word);

                    } else {

                        repeatingWords.add(word);

                    }

                }

                findRepeatingWordsDone.set(true);

                lock.notifyAll();

            }

        });

        Thread removeRepeatingWordsThread = new Thread(() -> {

            synchronized (lock) {

                while (!findRepeatingWordsDone.get()) {

                    try {

                        lock.wait();

                    } catch (InterruptedException e) {

                        e.printStackTrace();

                    }

                }

                for (String word : repeatingWords) {

                    sentence = sentence.replace(word, "");

                }

                removeRepeatingWordsDone.set(true);

                lock.notifyAll();

            }

        });

        readerThread.start();

        findRepeatingWordsThread.start();

        removeRepeatingWordsThread.start();

        try {

            readerThread.join();

            findRepeatingWordsThread.join();

            removeRepeatingWordsThread.join();

            if (removeRepeatingWordsDone.get()) {

                System.out.println("Resulting Sentence: " + sentence);

            } else {

                throw new NoRepeatingWordsException("No repeating words found in the sentence.");

            }

        } catch (InterruptedException | NoRepeatingWordsException e) {

            System.err.println("Error: " + e.getMessage());

        }

    }

}

Enter a sentence: hi my name name is avani

Repeating Word Found: name

Resulting Sentence: hi my name is avani

6

class OddArraySizeException extends Exception {

    public OddArraySizeException(String message) {

        super(message);

    }

}

 class ArraySearch {

    private static final int ARRAY\_SIZE = 10;

    private static int[] array = new int[ARRAY\_SIZE];

    private static final int ELEMENT\_TO\_FIND = 5;

    public static void main(String[] args) {

        try {

            if (ARRAY\_SIZE % 2 != 0) {

                throw new OddArraySizeException("Array size is odd.");

            }

            initializeArray();

            Thread firstThread = new Thread(() -> searchInFirstHalf(array, ELEMENT\_TO\_FIND));

            Thread secondThread = new Thread(() -> searchInSecondHalf(array, ELEMENT\_TO\_FIND));

            firstThread.start();

            secondThread.start();

            try {

                firstThread.join();

                secondThread.join();

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

            searchInArray(array, ELEMENT\_TO\_FIND);

        } catch (OddArraySizeException e) {

            System.err.println("Error: " + e.getMessage());

        }

    }

    private static void initializeArray() {

        for (int i = 0; i < ARRAY\_SIZE; i++) {

            array[i] = i;

        }

    }

    private static void searchInFirstHalf(int[] array, int target) {

        for (int i = 0; i < ARRAY\_SIZE / 2; i++) {

            if (array[i] == target) {

                System.out.println("Element " + target + " found in the first half.");

                return;

            }

        }

    }

    private static void searchInSecondHalf(int[] array, int target) {

        for (int i = ARRAY\_SIZE / 2; i < ARRAY\_SIZE; i++) {

            if (array[i] == target) {

                System.out.println("Element " + target + " found in the second half.");

                return;

            }

        }

    }

    private static void searchInArray(int[] array, int target) {

        for (int i = 0; i < ARRAY\_SIZE; i++) {

            if (array[i] == target) {

                System.out.println("Element " + target + " found in the entire array.");

                return;

            }

        }

        System.out.println("Element " + target + " not found in the entire array.");

    }

}

When array size is 10

Element 5 found in the second half.

Element 5 found in the entire array.

When array size is 7

Error: Array size is odd.

7

class InsufficientBalanceException extends Exception {

    public InsufficientBalanceException(String message) {

        super(message);

    }

}

class Banking {

    private double balance;

    public Banking(double initialBalance) {

        this.balance = initialBalance;

    }

    public void deposit(double amount) {

        balance += amount;

        System.out.println("Deposited: " + amount);

        System.out.println("Current Balance: " + balance);

    }

    public void withdraw(double amount) throws InsufficientBalanceException {

        if (amount > balance) {

            throw new InsufficientBalanceException("Insufficient balance. Withdrawal not allowed.");

        }

        balance -= amount;

        System.out.println("Withdrawn: " + amount);

        System.out.println("Current Balance: " + balance);

    }

    public double getBalance() {

        return balance;

    }

}

 class BankingApp {

    public static void main(String[] args) {

        Banking account = new Banking(1000.0); // Initial balance

        try {

            account.deposit(500.0);

            account.withdraw(3000.0); // This will trigger an exception

        } catch (InsufficientBalanceException e) {

            System.err.println("Error: " + e.getMessage());

        }

        System.out.println("Final Balance: " + account.getBalance());

    }

}

Deposited: 500.0

Current Balance: 1500.0

(when withdraw amt =3000)

Error: Insufficient balance. Withdrawal not allowed.

Final Balance: 1500.0

8

import java.util.Collection;

 class PrimeCount<T extends Number> {

    public int countPrimes(Collection<T> collection) {

        int primeCount = 0;

        for (T element : collection) {

            if (isPrime(element.intValue())) {

                primeCount++;

            }

        }

        return primeCount;

    }

    private boolean isPrime(int number) {

        if (number <= 1) {

            return false;

        }

        if (number <= 3) {

            return true;

        }

        if (number % 2 == 0 || number % 3 == 0) {

            return false;

        }

        for (int i = 5; i \* i <= number; i += 6) {

            if (number % i == 0 || number % (i + 2) == 0) {

                return false;

            }

        }

        return true;

    }

    public static void main(String[] args) {

        PrimeCount<Integer> primeCounter = new PrimeCount<>();

        // Example usage with a collection of integers

        java.util.ArrayList<Integer> integerList = new java.util.ArrayList<>();

        integerList.add(2);

        integerList.add(7);

        integerList.add(4);

        integerList.add(11);

        int primeCount = primeCounter.countPrimes(integerList);

        System.out.println("Number of prime numbers in the collection: " + primeCount);

    }

}

Number of prime numbers in the collection: 3

9

1st code

**Node<B>** is not considered a subtype of **Node<A**> even though **B** is a subtype of **A**. You cannot assign a **Node<B>** to a reference of type **Node<A>** directly.

2nd code

The code will compile without errors.

10