**JAVA TRAINING**

**DAY 1:**

**TOPICS:**

* Intro
* Compile basics
* Helloworld path
* Classpath
* Primitive types
* Primitive type conversion
* Operators

1. Intro to Java

Java is an object-oriented, platform-independent programming language that is widely used for developing applications. Key concepts include:

Syntax: Java has a C/C++ style syntax.

JVM, JDK, JRE: Understanding the role of the Java Virtual Machine (JVM), Java Development Kit (JDK), and Java Runtime Environment (JRE) is crucial.

2. Compile Basics

Compiling Java code involves using the javac command to turn .java source files into .class bytecode files that the JVM can execute.

Command:

javac MyProgram.java

java MyProgram

3. HelloWorld Path

Your first Java program is typically a simple "Hello, World!" to understand the structure of a Java application.

Example:

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

4. Classpath

Classpath is an environment variable that tells the JVM and javac where to look for user-defined classes and packages.

Example:

java -cp /path/to/classes MyClass

5. Primitive Types

Java has eight primitive data types:

byte, short, int, long: For integer values.

float, double: For floating-point numbers.

char: For characters.

boolean: For true/false values.

6. Primitive Type Conversion

Java supports type conversion between primitive types:

Implicit conversion (widening): smaller to larger data type, e.g., int to long.

Explicit conversion (narrowing): larger to smaller data type, e.g., double to int, using casting.

Example:

int num = 100;

double convertedNum = num; // implicit

double num2 = 9.99;

int intNum = (int) num2; // explicit (casting)

7. Operators Part One

Operators in Java include:

Arithmetic operators: +, -, \*, /, %.

Unary operators: ++, --, +, -.

Relational operators: ==, !=, >, <, >=, <=.

8. Operators Part Two

More operators:

Logical operators: &&, ||, !.

Bitwise operators: &, |, ^, ~, <<, >>, >>>.

Assignment operators: =, +=, -=, \*=, /=, etc.

Ternary operator: ? :.

**Practice Problems**

1. HelloWorld Path Problem:

Create a simple Java program called HelloJava.java that prints "Hello, Java!".

Compile and run it from the command line using javac and java.

2. Primitive Types Exercise:

Declare variables of all primitive types and print them.

Convert a double to an int and print the result.

3. Primitive Type Conversion Problem:

Write a program that takes a float as input, converts it to int, and displays both values.

Try a case where narrowing conversion loses precision (e.g., casting 123.456 to int).

4. Arithmetic Operators Problem:

Write a program that takes two integers from the user and performs all arithmetic operations, printing each result.

5. Logical and Relational Operators Problem:

Create a program that reads two integers and prints whether the first number is greater than the second using relational operators.

Use logical operators to combine multiple conditions.

6. Bitwise Operators Problem:

Implement a program that shifts an integer left and right by 2 positions and prints the result.

Use the ^ operator to demonstrate XOR functionality between two integers.

7. Ternary Operator Problem:

Write a program that reads an integer and prints "Even" if it is even, or "Odd" if it is odd, using a ternary operator.

**DAY 2:**

**TOPICS:**

* Operator part 3
* Program Control part 1
* Program Control part 2
* OOPS basics part 1
* OOPS basics part 2
* OOPS basics part 3
* “this” and Method Overloading
* Constructors

**Practice Problems:**

### **Operators (Part 3)**

1. **Arithmetic and Logical Operators:**
   * Write a program to find the factorial of a number using loops and arithmetic operators.
   * Create a program to check if a number is prime using logical operators.
2. **Bitwise Operators:**
   * Write a program to swap two numbers without using a third variable (use XOR).
   * Create a program to find the count of 1s in the binary representation of a number.
3. **Ternary Operator:**
   * Write a program to find the largest of three numbers using the ternary operator.

### **Program Control (Part 1 & 2)**

1. **Loops:**
   * Write a program to print a pyramid pattern using nested loops.
   * Create a program to reverse a given number.
2. **Decision Making:**
   * Write a program to determine if a given year is a leap year.
   * Create a program to simulate a simple calculator using switch-case.
3. **Jump Statements:**
   * Write a program to print all numbers from 1 to 100, skipping multiples of 5 (use continue).
   * Create a program to exit a loop if the user enters the number 0 (use break).

### **OOP Basics (Parts 1, 2, and 3)**

1. **Class and Objects:**
   * Create a class Car with attributes like brand, model, and price. Write methods to display details of a car and calculate the discounted price.
2. **Encapsulation:**
   * Write a class BankAccount with private attributes for accountNumber and balance. Provide public methods to deposit, withdraw, and check the balance.
3. **Inheritance:**
   * Create a class Animal with a method speak(). Derive classes Dog and Cat and override the speak() method to display appropriate sounds.
4. **Polymorphism:**
   * Implement method overloading in a class Calculator with methods to add integers, doubles, and arrays of numbers.

### **“this” Keyword and Method Overloading**

1. **this Keyword:**
   * Create a class Employee with attributes id, name, and salary. Use the this keyword to resolve naming conflicts between local variables and instance variables.
2. **Method Overloading:**
   * Write a program that has multiple overloaded methods for calculating the area of a rectangle, circle, and triangle.

### **Constructors**

1. **Parameterized Constructors:**
   * Create a class Student with attributes like name and age. Initialize these attributes using a parameterized constructor.
2. **Default Constructor:**
   * Create a class Book with default values for title and author. Write a default constructor to initialize these attributes.
3. **Constructor Overloading:**
   * Write a program with a class Account that has overloaded constructors to initialize accountNumber and balance in different ways.

### **Additional Challenges**

* Create a program to manage a library system using OOP concepts, including classes for Book, Member, and Librarian.
* Implement a simple banking system where users can open accounts, deposit money, withdraw money, and check balance using OOP.

**DAY 3:**

**TOPICS:**

* Static (video number 17)
* Arrays (video number 18)
* Inheritance part 1 (video number 19)
* Inheritance part 2 (video number 20)
* Polymorphism (video number 21)
* Abstract classes (video number 22)
* Interface part 1 (video number 23)
* Interface part 2 (video number 24)

### **Static**

1. **Static Variables and Methods:**
   * Write a program with a class Counter that has a static variable to count the number of objects created. Create three objects and print the count using a static method.
   * Create a class Utility with a static method convertToUpper(String str) that takes a string as input and returns it in uppercase. Test this method in the main function.

### **Arrays**

1. **Single-Dimensional Array:**
   * Write a program to find the largest and smallest numbers in an array.
   * Create a program to calculate the sum of all elements in an array.
2. **Multi-Dimensional Array:**
   * Write a program to add two 2D matrices.
   * Create a program to transpose a given matrix.

### **Inheritance Part 1**

1. **Basic Inheritance:**
   * Create a class Person with attributes name and age. Derive a class Student that adds rollNumber and marks. Write methods to display both Person and Student details.
   * Create a class Vehicle with a method start(). Derive a class Car that overrides the start() method. Write a program to demonstrate method overriding.

### **Inheritance Part 2**

1. **Multi-Level Inheritance:**
   * Create a class Shape with a method area(). Derive a class Rectangle with attributes length and breadth, and override the area() method. Further, derive a class Square from Rectangle and demonstrate multilevel inheritance.
2. **Hierarchical Inheritance:**
   * Create a base class Animal with a method sound(). Derive two classes Dog and Cat from Animal. Override the sound() method to print “Bark” for Dog and “Meow” for Cat.

### **Polymorphism**

1. **Compile-Time Polymorphism (Method Overloading):**
   * Create a class Calculator with overloaded methods add(int, int), add(double, double), and add(int, int, int). Test all methods.
2. **Run-Time Polymorphism (Method Overriding):**
   * Write a program with a base class Employee that has a method calculateSalary(). Derive two classes Manager and Developer, and override the calculateSalary() method in both to display different salaries.

### **Abstract Classes**

1. **Abstract Class Example:**
   * Create an abstract class Shape with an abstract method calculateArea(). Derive classes Circle and Rectangle and implement the calculateArea() method in both.
2. **Partial Implementation:**
   * Create an abstract class Appliance with methods turnOn() (abstract) and turnOff() (non-abstract). Derive a class Fan and implement turnOn().

### **Interface Part 1**

1. **Basic Interface Implementation:**
   * Create an interface Animal with methods eat() and sleep(). Implement this interface in classes Dog and Cat.
2. **Interface with Fields:**
   * Define an interface Constants with static fields like PI = 3.14. Create a class Circle that uses this constant to calculate the area of a circle.

### **Interface Part 2**

1. **Multiple Interfaces:**
   * Create two interfaces Printer with a method print() and Scanner with a method scan(). Create a class AllInOne that implements both interfaces.
2. **Interface Inheritance:**
   * Define an interface Vehicle with a method start(). Derive two interfaces FourWheeler and TwoWheeler from it, each adding its specific methods. Implement these in relevant classes.

**DAY 4:**

**TOPICS:**

* Re-visit the topics learned in previous days

**DAY 5:**

**TOPICS:**

* Java imports and packages(video number 26)
* Java Access Modifiers(video number 27)
* Autoboxing varargs(video number 28)
* Enum in java(video number 29)
* Generics(video number 30)

### **Java Imports and Packages (Video 26)**

1. **Create and Use Packages:**
   * Create a package shapes with two classes:
     + Circle with a method to calculate the area of a circle.
     + Rectangle with a method to calculate the area of a rectangle.
   * Write a Main class in the default package to import and use these classes.
2. **Static Imports:**
   * Use Math class methods like sqrt() and pow() in a program to calculate the hypotenuse of a right-angled triangle using static imports.

### **Java Access Modifiers (Video 27)**

1. **Access Modifier Example:**
   * Create a class Person with:
     + name (public)
     + age (protected)
     + address (default)
     + salary (private)
   * Create a subclass Employee in the same package to demonstrate which members are accessible.
   * Create another class TestAccess in a different package to test accessibility.

### **Autoboxing and Varargs (Video 28)**

1. **Autoboxing Example:**
   * Write a program to add a list of integers using a List<Integer> and demonstrate how autoboxing allows seamless addition of primitive int.
2. **Varargs Example:**
   * Create a method sum(int... numbers) that takes a variable number of integers as input and returns their sum. Test it with different numbers of arguments.

### 

### **Enum in Java (Video 29)**

1. **Basic Enum Usage:**
   * Define an enum Day with values for all days of the week.
   * Write a program to print a message like "It's a weekday" or "It's a weekend" based on the day.
2. **Enum with Methods:**
   * Create an enum Season with values WINTER, SPRING, SUMMER, and FALL. Add a method getAverageTemperature() to return an average temperature for each season.

### **Generics (Video 30)**

1. **Generic Class:**
   * Create a generic class Pair<K, V> to hold a key-value pair. Write a program to store and display pairs of different data types, like <String, Integer> or <Integer, String>.
2. **Generic Method:**
   * Write a generic method printArray(T[] array) to print elements of any array type. Test it with arrays of String, Integer, and Double.

### **Additional Quick Challenge**

* Create a mini-application combining these topics:
  + Use a package utilities containing a generic class Container<T> to store any object.
  + Use an enum Priority with values LOW, MEDIUM, and HIGH.
  + Use a Main class to demonstrate autoboxing, access modifiers, and package imports.

**DAY 6:**

**TOPICS:**

* Java Lang Objects (31 & 32)
* Clone(33)
* Java Lang Class(34)
* String(35)
* Exceptional Handling(Part One)(36)

**Java Lang Objects**

Create a program that demonstrates:

1. Creating an object of a class.
2. Using the basic methods of the Object class: getClass() and toString().

**Clone**

Create a class Student with fields for the name and age. Write a program to create a clone (copy) of a Student object using the clone() method.

**Java Lang Classes**

**Write a program to demonstrate the use of the following java.lang classes:**

1. **Math - Perform basic mathematical operations like finding the square root, power, and maximum of two numbers.**
2. **String - Manipulate a string by finding its length, converting it to uppercase, and extracting a substring.**
3. **System - Print a message to the console and measure the execution time of a simple loop using System.currentTimeMillis().**

**String**

1) Write a program that takes a sentence as input from the user and performs the following operations:

1. Count the number of words in the sentence.
2. Reverse the entire sentence.
3. Replace all vowels in the sentence with the character '\*'.

2) Write a program that:

1. Takes two strings as input from the user.
2. Compares them lexicographically using compareTo() method.
3. Checks if the two strings are equal (case-sensitive and case-insensitive).
4. Concatenates the two strings and prints the result.

**Exceptional Handling**

Write a program that:

1. Accepts two integers from the user as input.
2. Attempts to divide the first integer by the second.
3. Implements exception handling to manage the following scenarios:
   * Division by zero: Display an error message "Error: Division by zero is not allowed.".
   * Invalid input: Display an error message "Error: Please enter valid integers.".
   * Any other exceptions: Display "An unexpected error occurred.".

**Day 7:**

**Topics**

* **Exception (part two)(37)**
* **Reflection**
* **Threads - Intro**
* **Thread life cycle**
* **Volatile and memory**
* **Synchronised**
* **Object and method lock**

**Exception(part Two)**

Write a Java program that takes two numbers as input from the user and divides the first number by the second. Handle the following exceptions:

1. **ArithmeticException**: If the second number is zero, display an error message: *"Division by zero is not allowed."*
2. **InputMismatchException**: If the user enters non-numeric input, display an error message: *"Invalid input. Please enter numeric values."*
3. **General Exception**: For any other unexpected errors, display a generic error message: *"An error occurred: [error message]."*

**Reflection**

Write a Java program to demonstrate the use of reflection. Your program should:

1. Dynamically load a custom class (e.g., Person) using its fully qualified name.
2. Display the following details about the class:
   * All declared methods and their return types.
   * All declared fields and their types.
   * All constructors.
3. Dynamically invoke a method on an object of the class, provided the method name and arguments as input.

**Threads - Intro**

Write a Java program to demonstrate the use of multithreading. The program should:

1. Create two threads:
   * **Thread 1**: Print numbers from 1 to 10 with a 500ms delay between each number.
   * **Thread 2**: Print the square of numbers from 1 to 10 with a 700ms delay between each number.
2. Ensure both threads run concurrently.
3. Use Thread class or Runnable interface for thread implementation.

**Object and method locking**

Write a Java program to demonstrate object-level and method-level locking using multithreading. The program should:

1. Create a shared resource class (e.g., SharedPrinter) with two synchronized methods:
   * **Method 1**: Print a given string multiple times.
   * **Method 2**: Print a countdown from a given number.
2. Use two threads:
   * **Thread 1**: Calls the first synchronized method (print string).
   * **Thread 2**: Calls the second synchronized method (countdown).
3. Demonstrate the difference between:
   * **Object-Level Locking**: When both threads use the same instance of the shared resource.
   * **Class-Level Locking**: When both threads use a static synchronized method or block.
4. Show how the locks affect the execution order of the methods.

Extra Problems:

1.

Given an array of strings patterns and a string word, return *the* ***number*** *of strings in* patterns *that exist as a* ***substring*** *in* word.

A **substring** is a contiguous sequence of characters within a string.

**Example 1:**

**Input:** patterns = ["a","abc","bc","d"], word = "abc"

**Output:** 3

**Explanation:**

- "a" appears as a substring in "abc".

- "abc" appears as a substring in "abc".

- "bc" appears as a substring in "abc".

- "d" does not appear as a substring in "abc".

3 of the strings in patterns appear as a substring in word.

2.

For a string sequence, a string word is **k-repeating** if word concatenated k times is a substring of sequence. The word's **maximum k-repeating value** is the highest value k where word is k-repeating in sequence. If word is not a substring of sequence, word's maximum k-repeating value is 0.

Given strings sequence and word, return *the* ***maximum k-repeating value*** *of word in sequence*.

**Example 1:**

**Input:** sequence = "ababc", word = "ab"

**Output:** 2

**Explanation:** "abab" is a substring in "ababc".

**Example 2:**

**Input:** sequence = "ababc", word = "ba"

**Output:** 1

**Explanation:** "ba" is a substring in "ababc". "baba" is not a substring in "ababc".

3.

Given a string s, return true *if the* s *can be palindrome after deleting* ***at most one*** *character from it*.

**Example 1:**

**Input:** s = "aba"

**Output:** true

**Example 2:**

**Input:** s = "abca"

**Output:** true

**Explanation:** You could delete the character 'c'.

**Example 3:**

**Input:** s = "abc"

**Output:** false

4.

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

**Example 1:**

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1].

5.

We have n chips, where the position of the ith chip is position[i].

We need to move all the chips to **the same position**. In one step, we can change the position of the ith chip from position[i] to:

* position[i] + 2 or position[i] - 2 with cost = 0.
* position[i] + 1 or position[i] - 1 with cost = 1.

Return *the minimum cost* needed to move all the chips to the same position.

**Input:** position = [1,2,3]

**Output:** 1

**Explanation:** First step: Move the chip at position 3 to position 1 with cost = 0.

Second step: Move the chip at position 2 to position 1 with cost = 1.

Total cost is 1.

6.

There is a biker going on a road trip. The road trip consists of n + 1 points at different altitudes. The biker starts his trip on point 0 with altitude equal 0.

You are given an integer array gain of length n where gain[i] is the **net gain in altitude** between points i​​​​​​ and i + 1 for all (0 <= i < n). Return *the* ***highest altitude*** *of a point.*

**Example 1:**

**Input:** gain = [-5,1,5,0,-7]

**Output:** 1

**Explanation:** The altitudes are [0,-5,-4,1,1,-6]. The highest is 1.

7. <https://www.hackerrank.com/challenges/java-inheritance-2/problem?isFullScreen=true>

8. <https://www.hackerrank.com/challenges/java-interface/problem?isFullScreen=true>

**Note: for the hacker rank codes, Complete the code in the ide and copy the code and push it in the git repository.**

**Day 8:**

**Topics:**

* **Thread pool**
* **Countdown Latch**
* **Blockingqueue and wait**
* **Reentrantlock**
* **Arrays**

**Arrays - Problems:**

1. **Problem Statement:**

Given an array of integers, find the largest sum of a contiguous subarray. Implement an efficient algorithm to solve this problem.

Example:

Input: nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output: 6

Explanation: The subarray [4, -1, 2, 1] has the largest sum = 6.

1. Write a function to rotate an array of n integers to the right by k steps.

Example:

Input: nums = [1, 2, 3, 4, 5, 6, 7], k = 3

Output: [5, 6, 7, 1, 2, 3, 4]

Explanation: Rotate the array to the right by 3 steps.

1. **Problem Statement:**Given an array containing n distinct numbers in the range [0, n], find the one number that is missing from the array.

*Example:*

Input: nums = [3, 0, 1]

Output: 2

Explanation: The missing number is 2 because it is not in the array.

1. **Problem Statement:**

Given an array of integers and a target number, return the indices of the two numbers that add up to the target. Assume that there is exactly one solution.

Example:

Input: nums = [2, 7, 11, 15], target = 9

Output: [0, 1]

Explanation: nums[0] + nums[1] = 2 + 7 = 9.

1. **Problem Statement:**For each element in an array, find the next greater element to its right. If no such element exists, return -1 for that number.

*Example:*

Input: nums = [4, 5, 2, 10]

Output: [5, 10, 10, -1]

Explanation: The next greater element for 4 is 5, for 5 is 10, for 2 is 10, and for 10 there is no greater element.

**Day 9:**

**Topics:**

* **General Classes**
* **Vector and Hashtable**
* **How Hashtable works**
* **Collections Intro**
* **ArrayList**

### **Problem Statement: Vector**

**Student Grades Management System**

You are tasked with creating a program to manage student grades using a **Vector** in Java. Implement the following functionalities:

1. Add grades (as integers) to a vector dynamically.
2. Print all grades in the vector.
3. Calculate and display:
   * The highest grade.
   * The lowest grade.
   * The average grade.
4. Allow the user to remove a specific grade by its index.

**Input Example**:

* Add grades: 78, 85, 92, 67, 88.
* Remove grade at index 3.

**Output Example**:

Grades: [78, 85, 92, 88]

Highest Grade: 92

Lowest Grade: 78

Average Grade: 85.75

### **Problem Statement: Hashtable**

**Contact Directory Management System**

Create a program to manage a contact directory using a **Hashtable** in Java. Implement the following functionalities:

1. Add contacts with the name as the key and phone number as the value.
2. Search for a contact by name and display their phone number.
3. Remove a contact by name.
4. Display all contacts in the directory.

**Input Example**:

* Add contacts: John -> 9876543210, Alice -> 9123456780, Bob -> 8899776655.
* Search for Alice.
* Remove contact Bob.

**Output Example**:

Contact Directory:

John: 9876543210

Alice: 9123456780

Search Result for Alice: 9123456780

After Removal:

John: 9876543210

Alice: 9123456780

### **Problem Statement: ArrayList**

### **Problem Statement 1: Manage a Shopping List**

Create a program that allows the user to manage a shopping list using an **ArrayList**. Implement the following features:

1. Add items to the shopping list.
2. Remove an item from the list.
3. Display the shopping list.

**Input Example:**

* Add items: "Milk", "Eggs", "Bread", "Butter".
* Remove "Eggs".

**Output Example:**

Shopping List: [Milk, Bread, Butter]

### 

### **Problem Statement 2: Reverse the ArrayList**

Create a program that reverses the order of elements in an **ArrayList**.

**Input Example:**

* Input list: [1, 2, 3, 4, 5]

**Output Example:**

Reversed List: [5, 4, 3, 2, 1]

### **Problem Statement 3: Find the Maximum Element in an ArrayList**

Create a program to find the maximum element in an **ArrayList** of integers.

**Input Example:**

* Input list: [15, 22, 8, 42, 17]

**Output Example:**

Maximum Element: 42

**Day 10:**

**Topics**

* **LinkedList**
* **Set**
* **Map**
* **Stack and Queue**

**LinkedList:**

Create a program that allows the user to manage a task list using a LinkedList. Implement the following features:

1. Add a task to the end of the list.
2. Remove a task from the list by its name.
3. Display all tasks in the list in the order they were added.

**Input Example:**

* Add tasks: "Finish homework", "Go for a run", "Prepare dinner".
* Remove task: "Go for a run".

**Output Example:**

**Task List:**

**1. Finish homework**

**2. Prepare dinner**

**Set:**

Create a program that uses a Set to store a collection of unique names. Implement the following features:

1. Add names to the set.
2. Check if a name is already present in the set.
3. Display all unique names in the set.

**Input Example:**

* Add names: "Alice", "Bob", "Alice", "Eve", "Charlie".

**Output Example:**

**Unique Names: [Alice, Bob, Eve, Charlie]**

**Map:**

Create a program that uses a Map to store and manage student names and their corresponding grades. Implement the following features:

1. Add a student's name and grade to the map.
2. Update a student's grade if the student already exists in the map.
3. Retrieve a student's grade by their name.
4. Display all students and their grades.

**Input Example:**

* Add: {"Alice": 85, "Bob": 90, "Charlie": 88}
* Update: {"Bob": 92}
* Retrieve: "Alice"

**Output Example:**

**Alice's grade: 85**

**All students and grades:**

**Alice: 85**

**Bob: 92**

**Charlie: 88**

**Stack and Queue**

### **Problem Statement for Stack: Undo Functionality**

**Create a program that uses a Stack to implement an Undo feature for a text editor. The program should:**

1. **Allow the user to perform actions like typing words (e.g., "Hello", "World").**
2. **Store each action in the stack.**
3. **Undo the last action by popping it from the stack.**

**Input Example:**

* **Actions: "Type Hello", "Type World", "Undo", "Type Java".**

**Output Example:**

**Current Text: Hello Java**

### **Problem Statement for Queue: Customer Support System**

**Create a program that uses a Queue to manage a Customer Support System. The program should:**

1. **Add customers to the queue as they arrive.**
2. **Serve customers in the order they arrived (FIFO - First In, First Out).**
3. **Display the list of waiting customers.**

**Input Example:**

* **Add customers: "Alice", "Bob", "Charlie".**
* **Serve customer: 1.**

**Output Example:**

**Serving: Alice**

**Waiting: [Bob, Charlie]**

**Day 11:**

**Topics:**

* **Regex**
* **Concurrentmap**
* **Generics**
* **Java IO**
* **IO examples**

**Regex:**

### **Problem Statement**

Task: Write a Java program to validate a list of email addresses using regular expressions. The program should:

1. Accept a list of email addresses as input.
2. Validate each email address against the following rules:
   * It must start with a letter or a number.
   * It can include letters, numbers, dots (.), hyphens (-), and underscores (\_).
   * The domain name should be valid, ending with .com, .org, or .net.
3. Print whether each email address is valid or invalid.

### **Example Input**

**Input: ["user@example.com", "user.name@domain.org", "invalid-email@domain", "user@domain\_net", "@missinguser.com"]**

### **Expected Output**

**user@example.com: Valid**

**user.name@domain.org: Valid**

**invalid-email@domain: Invalid**

**user@domain\_net: Invalid**

**@missinguser.com: Invalid**

**Concurrentmap:**

### **Problem Statement: Online Voting System**

Create an Online Voting System using a ConcurrentMap where:

1. Each candidate's name is stored as a key in the ConcurrentMap, and their vote count is the value.  
   For example:  
   ConcurrentMap<String, Integer> votes = new ConcurrentHashMap<>();
2. Multiple threads simulate voters casting their votes for candidates concurrently.
3. The program should handle:
   * Adding a new candidate to the map if not already present.
   * Incrementing the vote count for an existing candidate safely.
4. At the end of voting, display the final vote counts for all candidates.

**Input Example (Simulated by Threads):**

* **Thread 1: Votes for "Alice"**
* **Thread 2: Votes for "Bob"**
* **Thread 3: Votes for "Alice"**
* **Thread 4: Adds a new candidate "Charlie" and votes for them.**

**Output Example:**

**Final Vote Counts:**

**Alice: 2**

**Bob: 1**

**Charlie: 1**

**Generics:**

Create a Generic Box class that can hold a single object of any type. The class should:

1. Be able to store any type of object, whether it's a String, Integer, Double, or any custom class.
2. Provide methods to:
   * setItem(T item): Store an item in the box.
   * getItem(): Retrieve the stored item.
   * printItem(): Print the item stored in the box.
3. Implement a method to check if the stored item is of a specific type, for example, using instanceof.

### Requirements:

1. The Box class should be generic to store any type of object.
2. The class should support methods to set, get, and print the item.
3. Implement a type check (isTypeOf) that returns true if the item is of a specified type.

**Day 12:**

**Topics:**

* **IO serialization**
* **Java nio**
* **Java net**
* **Sql intro**
* **Sql connection**
* **Sql statement**
* **Sql select**

### **Exercise 1: Creating a File and Writing Your Name**

**In this exercise, you will create a new file (e.g., name.txt) and write your name to it.**

**Task:**

* **Write a program that:**
  1. **Creates a new text file called name.txt.**
  2. **Write your name to the file.**

### **Exercise 2: Serialize and Deserialize a Java Object**

#### **Steps:**

1. **Create a class that implements the Serializable interface.**
2. **Serialize the object to a file using ObjectOutputStream.**
3. **Deserialize the object from the file using ObjectInputStream.**

### **Exercise 3: External Serializable a Java Object**

#### **Steps:**

1. **writeExternal(ObjectOutput out) for writing object data.**
2. **readExternal(ObjectInput in) for reading object data.**

### **Exercise 4: Java nio**

### **Steps:**

#### **Using any two buffers for Reading and Writing Data**

### **Exercise 5: Sql**

### **Steps:**

1. Create 3 students mark statement records using jdbc.

**Day -13**

**Topics:**

* **Java sql preparedstatement.**
* **Batching transaction metadata.**
* **Junit two and three.**

Task:

**Note - write at least 5 test cases for each method. (including Exceptions)**

1. **String Reversal Method**
   1. Write a method to reverse a string. Test edge cases like an empty string, a single character, and a string with special characters.
2. **Palindrome Checker**
   1. Implement a method to check if a string is a palindrome. Write test cases for valid palindromes, non-palindromes, and case sensitivity.
3. **Prime Number Checker**
   1. Write a method to check if a number is prime. Test cases should cover boundary values like 0, 1, negative numbers, and large primes.
4. **Factorial Calculator**
   1. Create a method to calculate the factorial of a number. Include test cases for zero, small positive numbers, and invalid inputs (e.g., negative numbers).
5. **Array Sorting**
   1. Write a method to sort an array of integers. Test cases should include arrays with duplicates, already sorted arrays, and empty arrays.

**Day -14**

**Topics:**

* **XML Intro**
* **XML Sax Parsing**
* **XML DOCS**
* **XML Jaxb**
* **JVM Internals**
* **Heap Memory**
* **GC**
* **Debugging**

Task:

1. Parse the XML and print the values

<?xml version="1.0" encoding="UTF-8"?>

<college>

<students>

<student>

<id>1</id>

<name>John Doe</name>

<department>Computer Science</department>

<grade>A</grade>

</student>

<student>

<id>2</id>

<name>Jane Smith</name>

<department>Electronics</department>

<grade>B</grade>

</student>

<student>

<id>3</id>

<name>Emma Brown</name>

<department>Mechanical</department>

<grade>A</grade>

</student>

</students>

</college>

1. Debug the program find bug and resolve it

public class DebuggingExercise {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of students: ");

int numStudents = scanner.nextInt();

int[] marks = new int[numStudents];

int totalMarks = 0;

// Collect marks

System.out.println("Enter marks for " + numStudents + " students:");

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

System.out.print("Student " + (i + 1) + ": ");

marks[i] = scanner.nextInt();

totalMarks += marks[i];

}

// Calculate average

double averageMarks = totalMarks / numStudents;

// Display results

System.out.println("Total Marks: " + totalMarks);

System.out.println("Average Marks: " + averageMarks);

}

}

**Day - 15**

**Topics:**

* **Runtime annotations**
* **Compile annotations**
* **Java default methods**
* **Inner class**
* **Lambda intro**
* **Lambda two**
* **Lambda three**

**Task:**

### **Sort a List of Students**

### Write a method that takes a list of Student objects and sorts them by their GPA in descending order using a lambda expression.

### **Find the Top Scorer in Each Department**

* 1. Write a method that groups students by department and finds the top scorer in each department using Collectors.groupingBy and maxBy.

### **Transform a List of Names**

* 1. Write a method that takes a list of student names and transforms each name into uppercase letters with a prefix like "Student: " using map.

### **Calculate Average GPA**

* 1. Write a method that calculates the average GPA of a list of students using a lambda expression.

### **Find Students with Names Starting with a Letter**

* 1. Write a method that finds all students whose names start with a specific letter.

**Day - 16**

**Topics:**

* **Lambda four**
* **Predicate and higherOrder function**
* **Predicate outBoxing**
* **Consumer**
* **Functions**

**Task :**

1. **Filter Even Numbers**Write a program to filter all even numbers from a list of integers using a Predicate. Use a higher-order function to perform the filtering.  
   Input: [12, 45, 22, 35, 60]  
   Output: [12, 22, 60]
2. **String Starts with Letter**Create a method that takes a Predicate<String> and a list of strings and filters out all strings that start with a particular letter (e.g., 'A').  
   Input: ["Apple", "Banana", "Avocado", "Cherry"]  
   Letter: 'A'  
   Output: ["Apple", "Avocado"]
3. **Students with Passing Grades**Write a function that accepts a Predicate<Student> to filter students with grades above 60. Define a Student class with fields like name and grade.  
   Input:
   1. John: 75
   2. Alice: 55
   3. Mark: 88  
      Output: John, Mark
4. **Modify Student Grades**Write a program that accepts a Consumer<Student> to increase each student's grade by 10. Define a Student class and modify a list of Student objects.  
   Input:
   1. John: 70
   2. Alice: 60

Output :

John: 80

Alice: 70

1. Create a BiConsumer<Integer, Integer> that takes two numbers and prints their sum.  
   Input:
   1. 3, 5

Output : Sum of 3 and 5 is 8

1. Use a Function<String, Integer> to map a list of strings to their lengths.
   1. **Input**: ["Hello", "World", "Java"]
   2. **Output**: [5, 5, 4]

**Day - 17**

**Topics:**

* **HigherOreder Function and Supplier**
* **Unary and binary operations**
* **Optinal**
* **Method Reference**
* **Method Reference - two**
* **Method Reference constructor**

**Task :**

1. **Simple Supplier** Write a Java program to use a Supplier that returns the current date and time.
2. **Random Supplier** Create a Supplier that generates a random number between 1 and 100 and prints it 5 times.
3. **UnaryOperator** Write a Java program to use a UnaryOperator that converts a given string to uppercase.
4. **BinaryOperator** Create a program using a BinaryOperator to find the maximum of two integers.
5. **Handling Null with Optional** Write a program that accepts a String input, converts it to uppercase if it's not null, and prints "Empty input" if the string is null.
6. **Using Optional with Default Values** Write a program that uses Optional to provide a default value of "Guest" when no name is provided.
7. **Static Method Reference** Write a Java program that uses a method reference to a static method Math::sqrt to calculate the square root of a list of numbers.
8. **Instance Method Reference** Create a program where a method reference is used to compare strings (e.g., String::compareToIgnoreCase) in a list and sort them.
9. **Method Reference Constructor** : Create a functional interface StudentFactory that contains a method create(String name, int marks). Use a constructor reference to implement this method to create Student objects.
10. Create a Student class with fields name and marks. Write an instance method displayStudentDetails in the Student class to print the student's details. Use a method reference to call this method for each student in a list.

**Day - 18**

**Topics:**

* [Stream - one](https://drive.google.com/drive/u/0/folders/10pcwPU-rZm3Anl_15ujkoRUBgvtEkpcz?usp=drive_copy)
* [Stream - two](https://drive.google.com/drive/u/0/folders/10pcwPU-rZm3Anl_15ujkoRUBgvtEkpcz?usp=drive_copy)
* [Stream Methods - two](https://drive.google.com/drive/u/0/folders/10pcwPU-rZm3Anl_15ujkoRUBgvtEkpcz?usp=drive_copy)
* [Stream Methods - two](https://drive.google.com/drive/u/0/folders/10pcwPU-rZm3Anl_15ujkoRUBgvtEkpcz?usp=drive_copy)
* [Stream Collectors](https://drive.google.com/drive/u/0/folders/10pcwPU-rZm3Anl_15ujkoRUBgvtEkpcz?usp=drive_copy)

Practice problems:

This project is about analyzing the performance of students in different departments using Java Streams. Here's the step-by-step explanation:

Imagine we have students in three departments (like Computer Science, Mathematics, and Physics). Each student has:

* A **name** (e.g., "John"),
* **Marks** for 5 subjects (e.g., 85, 90, 78, etc.),
* A **grade** (to be calculated based on their average marks).

We want to:

1. Find the **top 3 students** based on their average marks.
2. Assign a **grade** (A, B, C, D) to each student based on their average marks.
3. Find the **average marks** for each department.
4. Identify the **top student** in each department.
5. Count how many students **failed** in each department (average marks < 50).

**Day - 19**

**Topics:**

* [24 Fork join Pool](https://drive.google.com/open?id=12wpEEh6cqtcj3h1TkkGCgocBkWYxPcf8&usp=drive_copy)
* [25 Parallel Stream](https://drive.google.com/open?id=1_t_OIVCcbvvyxs2x0Cjr0xBX3hBAljtb&usp=drive_copy)
* [27 Callable and future](https://drive.google.com/open?id=1LPDJRuKKCTSrNfq5bd86Xf11Gy0GQvY9&usp=drive_copy)
* [26 executors and visual VM](https://drive.google.com/open?id=1GBoW2LoKtPwSvArkeup--W-ket6-Pigo&usp=drive_copy)
* [28 Completable and Future](https://drive.google.com/open?id=1xgthGh5UGhrv32kGZUB9fhi2U5jJOtol&usp=drive_copy)

Practice problems:

### **1. Fork/Join Pool**

**Exercise:**Write a program to calculate the **sum of integers from 1 to 1,000,000** using the **Fork/Join Pool**.

* Create a class SumTask that extends RecursiveTask<Long>.
* Split the range of numbers into smaller tasks using the ForkJoinPool.
* Combine the results from subtasks to get the total sum.

### **2. Parallel Stream**

**Exercise:**Given a list of integers, perform the following tasks using **parallel streams**:

1. Filter numbers greater than 50.
2. Map these numbers to their squares.
3. Print the first 10 squared numbers.

### **3. Callable and Future**

**Exercise:**Write a program to calculate the factorial of a number using **Callable** and **Future**.

* Create a FactorialTask class that implements Callable<Long>.
* Use an ExecutorService to submit the task.
* Retrieve the result using the Future.get() method.

### **4. Executors and VisualVM**

**Exercise:**Simulate a simple **multithreaded task executor**:

1. Create a fixed-thread pool of size 5 using Executors.newFixedThreadPool(5).
2. Submit 10 tasks (e.g., printing numbers from 1 to 10) and observe how tasks are distributed among threads.
3. Use **VisualVM** to monitor the thread usage and performance.

### **5. CompletableFuture**

**Exercise:**Write a program using **CompletableFuture** to simulate the following tasks:

1. Task 1: Fetch user data (simulate with a Thread.sleep()).
2. Task 2: Fetch user orders (simulate with a Thread.sleep()).
3. Combine the results of both tasks and print the user details along with the orders.

Use thenCombine() to merge the results from both tasks.

**Day - 20**

**Topics:**

* [29 Files and Date time](https://drive.google.com/open?id=1rMwjIpFlD_1FTnj0JXAdo6PpFR9TRNds&usp=drive_copy)
* [30 JAVA - 9 Intro](https://drive.google.com/open?id=1jl_8sFoxfoG1ddQB1QsBO-jSVCAOtSo6&usp=drive_copy)

Practice problems:

Use **Date and Time API**, **Factory Methods**, and **Streams** to build a simple scheduling system.

1. Create a class Event with fields name, dateTime, and duration.
2. Create a list of events using List.of().
3. Provide the following functionality:
   * Display all events.
   * Filter events based on a specific date.
   * Calculate the duration between events using Duration.between.

#### **Example Input:**

Events:

* "Meeting", 2025-01-28T10:00, 1 hour
* "Workshop", 2025-01-28T13:00, 2 hours

**Example Output:**

* Events on 2025-01-28:
  + "Meeting"
  + "Workshop"

—---------------------------------------------------------------------------------------------------------------------------------------

**Mini Project: Student Management System using Java 8 Features:**

## **Objective:**

Develop a console-based Student Management System (SMS) that allows users to manage student records using Core Java and Java 8 Features. The project should support CRUD operations (Create, Read, Update, Delete) without using a database.

## **Problem Statement:**

You are required to build a **Student Management System** where users can:

1. **Add** new students with details like ID, Name, Age, Grade, and Email.
2. **View** all students.
3. **Search** for a student by ID.
4. **Update** student details.
5. **Delete** a student record.
6. **Store and retrieve** student records from a file (without using a database).

## **Technical Requirements:**

### **1️⃣ Core Java Concepts to Use:**

* Object-Oriented Programming (**OOP**): Create a Student class.
* Collections Framework (**ArrayList, HashMap, etc.**) for storing student records.
* Exception Handling for invalid inputs.
* File Handling for storing and retrieving data.

### **2️⃣ Java 8 Features to Implement:**

* **Streams & Lambda Expressions** for filtering and sorting students.
* **Method References** for clean code.
* **Optional Class** to avoid NullPointerException.
* **Functional Interfaces** for flexible operations.
* **Multithreading (Optional Bonus)**: Save student data to a file in a background thread.

## **Project Constraints:**

✅ The application must run in a **console/CLI** environment.  
✅ The data must be **stored and retrieved** using a file (**JSON, CSV, or text-based storage**).  
✅ You are **not allowed to use a database**.  
✅ The application should handle **invalid inputs gracefully**.

## **Bonus Features (Optional, for Extra Points 💡)**

🔹 Sort students by name or age using Java 8 Streams.  
🔹 Implement **search filters** (e.g., students with Grade A).  
🔹 Create a simple **menu-based CLI system** for better user experience.  
🔹 Use **multithreading** to save data in the background.

## **Expected Deliverables:**

📌 **Working Java Code** with proper modularity and clean code.  
📌 **File-based storage implementation** (Text, JSON, or CSV).  
📌 **README file** explaining how to run the project.

⏳ **Time Limit: 2 Days**

**Tip:** Use **your own coding style and approach!** There’s no single correct way to solve this problem—experiment and innovate. 🚀

## **Evaluation Criteria**

1. **Code Quality & Best Practices** (20%)
2. **Correct Implementation of Features** (30%)
3. **Usage of Java 8 Features** (20%)
4. **Error Handling & Edge Cases** (20%)
5. **Bonus: Multithreading & Advanced Features** (10%)

—---------------------------------------------------------------------------------------------------------------------------------------