Giri's Tech Hub Pvt.Ltd, Pune.

Programming (Machine) Test

Batch: Feb/March/May-24

Date: 12/12/2024
Time: 08:00 to 11:00 Am

Instructions:

1. **Solve any 8 questions.**
2. **Input should be from user.**
3. **Indentation and comments mandatory.**

Q1. Write a program that identifies and extracts all palindrome strings from a given text. A palindrome is a word that reads the same backward as forward (e.g., "radar", "level"). The program should include the following functionality:

Input "Wow, this is a radar and level test. Madam!"
Output: ['wow', 'radar', 'level', 'madam']

Constraints:

-Palindromes must have at least two characters.
-Ignore case sensitivity (e.g., "Level" and "level" should be treated as the same palindrome).
-Consider only alphanumeric characters; ignore punctuation and special symbols.

Q2. Design a Java class named `Employee` to manage employee details and apply performance-based bonuses. The program should include the following:

Attributes:
`id` (int): A unique identifier for the employee.
`name` (String): The name of the employee.
`designation` (String): The job title of the employee.
`salary` (double): The base salary of the employee.

Methods:
Constructor: Initialize the attributes `id`, `name`, `designation`, and `salary`.
`displayDetails`: Print the employee's details in a formatted manner.
`applyBonus`: Apply a bonus to the employee's salary based on performance rating:
`'excellent'`: 20% bonus
`'good'`: 10% bonus
`'average'`: 5% bonus
If the rating is invalid, print an error message and do not apply a bonus.

Functionality:
Create an `Employee` instance with the specified attributes.
Use methods to display details and apply bonuses based on the provided performance rating.

Q3. Design a Java program with the following requirements:
Book Class:
Attributes:
title (String): The title of the book.
author (String): The author of the book.
isbn (String): The International Standard Book Number (ISBN) of the book.

Methods:
Constructor to initialize the attributes.
Getters for all attributes.
Override toString() to provide a string representation of the book.

Library Class:
Manages a collection of Book objects.

Methods:
addBook(Book book): Add a book to the library.
searchByTitle(String title): Return a list of books matching the title (case-insensitive).
searchByAuthor(String author): Return a list of books by the specified author (case-insensitive).
displayBooks(): Display all books in the library. If no books exist, print a message indicating the library is empty.

Q4. Create a Java program to demonstrate exception handling by catching and handling multiple exceptions. The program should:

Functionality:
Accept a string array as input.
Parse the first element of the array to an integer.
Access an index in the array specified by the parsed integer.

Error Handling:
Catch and handle the following exceptions:
`ArrayIndexOutOfBoundsException`: Handle cases where the specified index is outside the bounds of the array.
`NumberFormatException`: Handle cases where the first element cannot be parsed into an integer.
Output:
Display custom error messages for each exception.
Handle cases gracefully to ensure the program does not crash.

Q5. Develop a Java program to handle file operations, including writing and reading data, and counting the occurrences of a specific word in the file. The program should:

Write Data:
Create a text file.
Write a given set of lines or text into the file.
Read Data:
Read the file content line by line.
Count Word Occurrences:
Accept a specific word as input.
Count and display the total occurrences of the word in the file, case-insensitively.
Error Handling:
Handle exceptions such as IOException in case of file-related errors.

Q6. Create a function that accepts a collection of strings (e.g., a list or any other iterable) and sorts the strings in lexicographical (alphabetical) order. The function should return the sorted collection.

Requirements:
Input: The function will accept a collection of strings (e.g., a list of strings).
Output: The function should return the collection of strings sorted in lexicographical order (which is akin to alphabetical order).
The sorting should be case-sensitive (i.e., uppercase letters come before lowercase letters).
Example input strings = ["banana", "Apple", "grape", "Cherry"] # Example output sorted_strings = ["Apple", "Cherry", "banana", "grape"]
Note:- Lexicographically means in a way that relates to dictionaries or is sorted using an algorithm based on alphabetical order.

Q7. Create a generic class Box that can store any type of object. The class should provide methods to:

Set an object of any type.

Get the stored object.

The class should be flexible to handle various data types, leveraging generics to ensure type safety.

Requirements:

Generic: The Box class should be able to store any type of object (e.g., integers, strings, custom objects, etc.).

Set Method: The class should provide a method to store an object inside the box.

Get Method: The class should provide a method to retrieve the stored object.

Type Safety: The class should maintain the type of the object it stores.

Q8. Write a program that demonstrates the use of a ListIterator to traverse and modify a list in both directions. The program should:

Traverse the list from left to right (forward direction).

Traverse the list from right to left (backward direction).

Modify elements during the traversal (e.g., changing values at specific positions).

Requirements:

Use ListIterator to traverse the list in both directions.

Demonstrate the ability to modify the list during traversal.

Include at least one modification while traversing forward and one modification while traversing backward.

The program should work with a list of integers or strings, but can be generalized for any type of list.

Q9. Write a program that creates a TreeMap (or an equivalent sorted map) to store employee information. In this map:

The key is the employee's ID (an integer).

The value is an Employee object containing details like name and position.

The program should:

Store employee information using the TreeMap (or a similar structure).

Ensure the map is sorted by employee ID in ascending order.

Display the list of employees in ascending order of their IDs.

Requirements:

Employee Class: Define an Employee class with at least the following properties:

id: Integer (employee's unique identifier).

name: String (employee's name).

position: String (employee's position in the company).

TreeMap: Use a sorted map to store the employees. In Python, this can be done using sortedcontainers.SortedDict (or dict in combination with sorting).

Sorting: The employees should be stored and displayed in ascending order of their id.

Display: Print the employee details (ID, name, and position) in ascending order of their ID.

Implement a polymorphic Bank Account system where there are two types of accounts:

SavingsAccount

CurrentAccount

Both account types should share common functionality such as deposit and withdrawal, but with some specific differences. Specifically, the CurrentAccount should impose a transaction fee on each withdrawal.

Implement method overriding and method overloading to handle various transaction types (like depositing and withdrawing amounts in different ways).

Requirements:

Base Class: Create a BankAccount class that contains common methods:

deposit(amount: float): Method to deposit money into the account.

withdraw(amount: float): Method to withdraw money from the account.

SavingsAccount Class: A derived class that represents a savings account. It should:

Inherit from BankAccount.

Override the withdraw method to simply deduct the specified amount.
CurrentAccount Class: A derived class that represents a current account. It should:
Inherit from BankAccount.

Override the withdraw method to apply a transaction fee on each withdrawal.
Method Overloading: The system should allow multiple ways of depositing or withdrawing money. For example:
You should be able to withdraw an amount and specify a fee type, or withdraw just the amount without specifying a fee.
Polymorphism: The program should use polymorphism to handle different account types uniformly. This can be done by referring to both SavingsAccount and CurrentAccount using a BankAccount reference.

Q10. Design an interface named Employee that will be implemented by two classes: Manager and Developer. Each class should have specific responsibilities and a unique salary calculation method.

Requirements:
Employee Interface:
Methods:
void performTask(): Each class should implement this method to represent the specific task the employee performs.
double calculateSalary(): Each class should implement this method to calculate the salary based on their respective attributes.

Manager Class:
Attributes:
name (String): The manager's name.
baseSalary (double): The manager's base salary.
teamSize (int): Number of team members managed by the manager.

Methods:
performTask(): Prints that the manager is managing a team.
calculateSalary(): Calculates salary as baseSalary + (teamSize * 1000).

Developer Class:
Attributes:
name (String): The developer's name.
baseSalary (double): The developer's base salary.
completedTasks (int): The number of tasks completed by the developer.

Methods:
performTask(): Prints that the developer is coding.
calculateSalary(): Calculates salary as baseSalary + (completedTasks * 200).

Q11. Create a Course Management System that manages courses, departments, and students' grades using a nested HashMap and ArrayList. The system should allow adding courses, enrolling students, retrieving student grades, and calculating the average grade for each course within a department.
Requirements:
Data Structure:
Outer HashMap: The key is the course name (String), and the value is another HashMap.
Inner HashMap: The key is the department name (String), and the value is an ArrayList of Student objects.

ArrayList of Student Objects: Each student has:
String name (the student’s name)
double grade (the student’s grade)

Operations:
Add Course: Add a course to the system, specifying the department for that course.
Enroll Student: Enroll a student in a specific course and department with a grade.
Get Grade: Retrieve the grade of a student in a specific course and department.
Calculate Average Grade: Calculate the average grade of all students enrolled in a specific course in a department.

Student Class:
Attributes:
String name (student's name)
double grade (student's grade)
Constraints:
Handle cases where courses or departments do not exist.
If a student is already enrolled in a course under a department, the system should handle the case accordingly (e.g., either ignore or overwrite).

Q12. Create a Ticket Booking System for a theater that allows multiple users to book tickets simultaneously. The system should ensure that the ticket availability is updated correctly when multiple users try to book tickets at the same time, preventing double booking.

Requirements:
TicketBooking Class:
Attributes:
`availableTickets`: Integer (The number of available tickets for the show).
Methods:
`bookTicket()`: A method that books a ticket by decreasing the number of available tickets. This method should be thread-safe to avoid double-booking.
`getAvailableTickets()`: A method that returns the current number of available tickets.
BookingThread Class:
This class simulates a ticket booking request.
It will create multiple threads, where each thread attempts to book tickets by calling the `bookTicket()` method.

Multithreading Operations:
Use threads to simulate multiple users attempting to book tickets at the same time.
Ensure that no more tickets are booked than the available tickets.
The system should prevent race conditions when multiple threads try to book the same ticket.
Thread Safety:
The `bookTicket()` method must ensure that only one thread can book a ticket at a time, using synchronization to prevent race conditions.

-----ALL THE BEST-----