Smart Cliff Learning Solutions

KIOT –DIV – Core Java– Technical Assessment

Module	**:**	**Core Java**
Date	**:**	**11/08/2025**
Time	**:**	**2:00pm to 4:30pm**
Duration	**:**	**2 : 30 Hrs.**
Total Marks	**:**	**50 Marks**

PART A: MCQ (20 x 1 : 20 Marks)

PART B: PROGRAMMING (30 MARKS)

1. Armstrong Number (5 marks)

A software company is developing a secure digital locker system. As part of the security algorithm, the locker's passcode is considered valid only if it is an Armstrong number.

An Armstrong number is a number where the sum of its digits, each raised to the power of the number of digits, equals the number itself.

For example:

- 153 → $1^3 + 5^3 + 3^3 = 153$ (valid)
- 123 → $1^3 + 2^3 + 3^3 = 36$ (invalid)

Task:

Write a program that **accepts a passcode (integer)** from the user and determines whether it is a valid Armstrong number. If valid, print **"Access Granted"**; **otherwise**, print **"Access Denied"**.

Sample Input and Output 1

Enter passcode: 153

Passcode is a valid Armstrong number.

Access Granted.

Sample Input and Output 2

Enter passcode: 123

Passcode is not a valid Armstrong number.

Access Denied.

2. Abstraction (10 Marks)

HCL Technologies hires employees across various categories based on their work requirements. The organization employs:

1. **Full-Time Employees** – Permanent staff members who receive a fixed base salary plus additional benefits.
2. **Part-Time Employees** – Employees who are paid based on the number of hours worked and an hourly rate.

The company wants to automate salary calculation for each employee type using Object-Oriented Programming and runtime polymorphism.

Requirements:

1. **Create a base class** Employee

- Contains employeeName, employeeID as common attributes.
- Declare an abstract method **calculateSalary ()** that will be overridden by subclasses.

2. Create subclasses **for each employee type:**

- **FullTimeEmployee** – Implement calculateSalary () as:

  salary = baseSalary + benefits
- **PartTimeEmployee** – Implement calculateSalary () as:

  salary = hourlyRate × hoursWorked

3. In the main program:

- Create objects for each employee type.
- Accept required input values from the user.
- Use **runtime polymorphism** to invoke calculateSalary () for each employee type.
- Display the calculated salary with proper labels.

Sample Input

Enter Full-Time Employee details:

Name: Rajesh

Base Salary: 40000

Benefits: 8000

Enter Part-Time Employee details:

Name: Priya

Hourly Rate: 500

Hours Worked: 40

Sample Output

Salary of Full-Time Employee Rajesh: 48000.0

Salary of Part-Time Employee Priya: 20000.0

3. Exception Handling (8 Marks)

HCL assigns employees to specific projects based on predefined project codes. Only the following project codes are valid: **P101**, **P102**, and **P103**.

Write a Java program that:

1. Accepts the **employee's name** and **project code** as input.
2. Validates the project code against the list of valid codes.
3. Throws a **custom exception,** "InvalidProjectCodeException" if the project code entered is not valid.
4. If the project code is valid, display the message:
   Employee <name> successfully assigned to project <code>.

Sample Input 1

Enter Employee Name: Rajesh

Enter Project Code: P102

Sample Output 1

Employee Rajesh successfully assigned to project P102.

Sample Input2

Enter Employee Name: Meena

Enter Project Code: P105

Sample Output2

Invalid project code! Please choose from P101, P102, P103.

4. Collections (7 Marks)

HCL wants to keep track of employee IDs registering for a training program.

Write a Java program that:

1. Accepts n employee IDs (integers) from the user and stores them in an **ArrayList<Integer>** in the order entered.
2. Removes duplicate employee IDs so that each ID appears only once.
3. Displays:
   - The original list of employee IDs (with duplicates)
   - The list after removing duplicates
   - The total count of unique employee IDs

Sample Input

Enter number of employees: 7
Enter employee ID 1: 101
Enter employee ID 2: 102
Enter employee ID 3: 101
Enter employee ID 4: 103
Enter employee ID 5: 102
Enter employee ID 6: 104
Enter employee ID 7: 103

Sample Output

Original employee IDs:
[101, 102, 101, 103, 102, 104, 103]

Employee IDs after removing duplicates:
[101, 102, 103, 104]

Total unique employee IDs: 4