Java Assessment Question Bank

1. Problem Statement: Student Course Management System

Project Overview

You are tasked with building a command-line Java application that simulates a basic student management system. This project will help you apply key concepts from Unit 2, including:

- **Object-Oriented Programming (OOP)**: Design and implement classes with encapsulated data and behavior.
- **Collections**: Use appropriate data structures (`ArrayList`, `HashMap`) to manage student records.
- **Exception Handling**: Use `try-catch` blocks to handle invalid input and logical errors.

Functional Requirements

Your application must include:

1. Student Class

Fields:

- `studentId` (String): Unique identifier for each student
- `name` (String): Student's full name
- `grade` (String): Current grade or class level
- `courses` (`List<String>`): List of enrolled course names

Methods:

- Constructor
- Getters and setters
- `enrollCourse(String courseName)`

- `dropCourse(String courseName)`
- `toString()` to display student details

2. StudentManager Class

Fields:

- `students` (`HashMap<String, Student>`): Stores all student records

Methods:

- `addStudent(Student student)`
- `removeStudent(String studentId)`
- `searchStudent(String studentId)`
- `enrollCourse(String studentId, String courseName)`
- `dropCourse(String studentId, String courseName)`
- `viewAllStudents()`

3. Main Class

Implements a menu-driven interface using `Scanner` **Options**:

- Add a student
- Remove a student
- Search student by ID
- Enroll student in a course
- Drop student from a course
- View all students
- Exit

2. Problem Statement: Employee Attendance Tracker

Project Overview

You are tasked with building a command-line Java application that simulates a basic employee attendance tracker. This project will help you apply key concepts from Unit 2, including:

- **Object-Oriented Programming (OOP)**: Design and implement classes with encapsulated data and behavior.
- **Collections**: Use appropriate data structures (`ArrayList`, `HashMap`) to manage employees and attendance records.
- **Exception Handling**: Use `try-catch` blocks to handle invalid input and logical errors.

Functional Requirements

Your application must include:

1. Employee Class

Fields:

- `employeeId` (String): Unique identifier for each employee
- `name` (String): Employee's full name
- `department` (String): Department name
- `attendanceDates` (`List<LocalDate>`): Dates on which the employee was marked present

Methods:

- Constructor
- Getters and setters
- `markAttendance(LocalDate date)`
- `getAttendanceCount()`
- `toString()` to display employee details

2. AttendanceManager Class

Fields:

- `employees` (`HashMap<String, Employee>`): Stores all employee records
- `presentToday` (`List<String>`): List of employee IDs marked present for the current day

Methods:

- `addEmployee(Employee employee)`

- `markAttendance(String employeeId)`
- `viewAttendance(String employeeId)`
- `viewAllEmployees()`
- `getSummary()` — shows total present/absent count for today

3. Main Class

Implements a menu-driven interface using `Scanner`**Options**:

- Add an employee
- Mark attendance
- View attendance by employee ID
- View all employees
- View today's attendance summary
-

3. Problem Statement: Expense Tracker Application

Project Overview

You are tasked with building a command-line Java application that simulates a simple expense tracker. This project will help you apply key concepts from Unit 2, including:

- **Object-Oriented Programming (OOP)**: Design and implement classes with encapsulated data and behavior.
- **Collections**: Use `ArrayList` to manage expense entries.
- **Exception Handling**: Use `try-catch` blocks to handle invalid input and runtime errors.

Functional Requirements

Your application must include:

1. Expense Class

Fields:

- `expenseId` (int): Unique identifier for each expense
- `amount` (double): Amount spent
- `category` (String): Expense category (e.g., Food, Travel)
- `description` (String): Optional notes

Methods:

- Constructor
- Getters and setters
- `toString()` to display expense details

2. ExpenseManager Class

Fields:

- `expenses` (`ArrayList<Expense>`): Stores all expense entries
- `nextId` (int): Tracks the next available expense ID

Methods:

- `addExpense(double amount, String category, String description)`
- `deleteExpense(int expenseId)`
- `viewAllExpenses()`
- `searchByCategory(String category)`
- `getTotalExpense()`

3. Main Class

Implements a menu-driven interface using `Scanner` **Options**:

- Add an expense
- Delete an expense
- View all expenses
- Search by category
- View total expense
- Exit