Package Assignment

1. Creating and Using User-Defined Packages

Objective: Practice creating and importing packages.
Problem Statement:
Create two packages:

- `college.student`
- `college.faculty`

1. In the `college.student` package, create a class `Student` with methods to display student name and roll number.
2. In the `college.faculty` package, create a class `Faculty` with methods to display faculty name and subject.
3. In the main class (not in any package), import both packages and call their respective methods to display details.

2. Package Hierarchy

Objective: Understand sub-packages and file organization.
Problem Statement:
Create a nested package structure:

```
com.university.department.cse
```

1. Create a class `Course` inside the above package to print course details.
2. Write another class `MainApp` (in the default package) that imports and uses the `Course` class.
3. Explain how the folder structure should look when you compile and run the program.

3. Static Import Demonstration

Objective: Understand `static import` feature.
Problem Statement:
Write a Java program that:

1. Uses `import static java.lang.Math.*;`
2. Demonstrates at least **five** static methods (e.g., `sqrt()`, `pow()`, `max()`, `min()`, `abs()`).
3. Explain why static import can be useful and when it should be avoided.

4. Modular Programming (Java 9+)

Objective: Introduce modular concepts.
Problem Statement:
Create a modular project with:

- Module 1: `collegeinfo` → contains a package `college.student` and class `Student`.
- Module 2: `app` → contains a class `MainApp` that uses `Student` from the first module.

Tasks:

1. Write the `module-info.java` file for both modules.
2. Demonstrate how to compile and run a modular application using the `--module-path` option.
3. Explain the difference between packages and modules.

5. Real-Life Scenario

Objective: Apply package concepts to a real-world use case.
Problem Statement:
Design a small application for a **Library Management System** using packages:

- `library.books` → Class `Book` (book details)
- `library.members` → Class `Member` (member details)
- `library.transactions` → Class `Transaction` (issue/return details)

In the main program, import all packages and simulate:

1. Adding a new book.
2. Registering a new member.
3. Issuing a book to a member.

6. Employee Management System (Multi-Package Project)

Objective: Implement modular code using multiple user-defined packages.

Packages to Create:

- `com.company.hr` – contains `Employee` class with fields: `id`, `name`, `department`, `salary`.
- `com.company.payroll` – contains `Payroll` class with a method `calculateBonus(Employee e)` that adds 10% bonus to salary.
- `com.company.main` – contains `MainApp` class that uses the other two packages.

Tasks:

1. Define all classes properly with encapsulation (private variables + getters/setters).
2. Import necessary classes using both **single** and **on-demand imports**.
3. Display employee details and calculated salary after bonus.
4. Show folder structure and explain why it must match the package declaration.

6. Student Performance Analyzer

Objective: Apply package and modular design concepts for analytics.

Packages:

- `com.school.data` – contains class `Student` with marks in 3 subjects.
- `com.school.util` – contains class `Analyzer` with methods:
  - `calculateAverage(Student s)`
  - `findGrade(double average)`
- `com.school.main` – main class to test functionality.

Tasks:

1. Use `import com.school.data.*;` and `import com.school.util.*;`
2. Display student name, marks, average, and grade.
3. Add a `toString()` method in `Student` to print details neatly.

7. Banking System with Static Import

Objective: Demonstrate the use of static import with Math library.

Packages:

- `com.bank.util` – class `InterestCalculator`
  - Methods: `calculateSimpleInterest()`, `calculateCompoundInterest()`.

In main program, use:

```
import static java.lang.Math.*;
```

- to use `pow()` for compound interest calculation.

Tasks:

1. Create methods using formulae:
   - SI = (P × R × T) / 100
   - CI = P × (pow((1 + R/100), T)) - P
2. Demonstrate both calculations using static import.

8. Package Access Modifier Control

Objective: Explore accessibility rules across packages.

Packages:

- `com.access.one` – class `Base` with methods using all 4 access modifiers.
- `com.access.two` – class `Derived` that extends `Base`.

Tasks:

1. Demonstrate which methods are accessible in the subclass from another package.
2. Print results and justify why others aren't accessible.
3. Write a table summarizing your findings (similar to access modifier table).

9. Company Analytics (With Sub-Packages)

Objective: Practice sub-packages and modular folder organization.

Package Hierarchy:

```
com.company.analytics.sales
com.company.analytics.hr
```

Tasks:

1. `sales` → class `SalesReport` prints region-wise sales data.
2. `hr` → class `EmployeeReport` prints employee performance data.
3. Main program imports both and prints a combined company report.
4. Explain how sub-packages are independent of parent packages in Java.

10. Mini Project – College Management System

Objective: Combine all learned concepts (packages, import, static import, modularity).

Packages:

`college.student`

`college.faculty`

`college.department`

`college.main`

Tasks:

1. Create `Student`, `Faculty`, and `Department` classes with relevant data and methods.
2. Use `import` and `static import` effectively.
3. Maintain proper folder structure for all packages.
4. In the `MainApp` class:
   - Create objects of each class.
   - Display complete college info (students, faculties, and departments).
5. Explain how packages helped in organizing this project.