Project Statement: Campus Course & Records Manager (CCRM)

1) Project Brief (what you'll build)

Design and implement a **console-based Java application** called **Campus Course & Records Manager (CCRM)** that lets an institute manage:

- **Students** (create/update, enroll/unenroll in courses)
- **Courses** (create/update, list, search, assign instructors)
- **Grades & Transcripts** (record marks, compute GPA, print transcripts)
- **File Utilities** (import/export CSV/JSON-like plain files, archive/backup course data)

This is a **Java SE** project built and run locally. It must be structured with clear **packages**, demonstrate **OOP** (Encapsulation, Inheritance, Abstraction, Polymorphism), robust **exception handling**, modern **I/O with NIO.2**, **Streams**, **Date/Time API**, and include **interfaces**, **abstract classes**, **nested classes**, **enums**, **lambdas**, **recursion**, and **design patterns** (Singleton & Builder).

> **Expected output on submission:**
> A **Git repository link** containing your complete project with a **README.md**, **screenshots**, and an **optional demo video** link.

---

2) Functional Requirements

1. **Student Management**
    - Add/list/update/deactivate students.
    - Each student has: `id`, `regNo`, `fullName`, `email`, `status`, `enrolledCourses`, and date fields using **Java Date/Time API**.
    - Print a **student profile** and **transcript**.

2. **Course Management**
    - Add/list/update/deactivate courses.
    - Each course has: `code`, `title`, `credits`, `instructor`, `semester`, and `department`.
    - Provide **search & filter** (by instructor, department, semester) using the **Stream API**.
3. **Enrollment & Grading**
    - Enroll/unenroll students to courses (with business rules, e.g., max credits per semester).
    - Record marks & compute letter **grades** and **GPA**.
    - **Enum** for `Semester` (e.g., SPRING, SUMMER, FALL) and `Grade` (S, A, B, …, F with grade points).
    - Generate a **transcript** view that uses **toString()** overrides and **polymorphism**.
4. **File Operations (NIO.2 + Streams)**
    - **Import** students/courses from simple CSV-like text files.
    - **Export** current data (students, courses, enrollments) to files.
    - **Backup** command that copies exported files to a timestamped folder (use `Path`, `Files`, `walk`, `copy`, `move`, `exists`).
    - A **recursive** utility (e.g., recursively compute and print total size of the backup directory, or recursively list files by depth).
5. **CLI Workflow**
    - Menu-driven console with options for all operations.
    - Use **switch** (classic or enhanced) and all **decision/loop constructs** (while/do-while/for/enhanced-for; demonstrate `break`, `continue`, and a labeled jump once).

---

3) Mandatory Technical Requirements

Setup & Platform

- **README** must include:
    - **Evolution of Java** (short timeline bullet points).
    - **Differentiate Java ME vs Java SE vs Java EE** (table or bullets).
    - **Java architecture**: JDK, JRE, JVM (what they are & how they interact).
    - **Install & configure Java on Windows** (steps + your **screenshot**).
    - **Using Eclipse IDE**: new project creation, run configs (**screenshots**).

Language & Core

- Show **Java syntax & structure** with a clearly defined `main` class.
- Organize code with **packages** (e.g., `edu.ccrm.domain`, `edu.ccrm.service`, `edu.ccrm.io`, `edu.ccrm.util`, `edu.ccrm.cli`).
- Demonstrate:
    - **Primitive variables**, **objects**, **operators** (arithmetic, relational, logical, bitwise), **operator precedence** (document a small example in code comments/tests).
    - **Decision structures**: `if / if-else / nested if-else`, and a **switch** menu.
    - **Loops**: `while`, `do-while`, `for`, **enhanced for**; include at least one **jump control** (`break`, `continue`).
    - **Arrays** and **Array utilities** (`Arrays` class): sorting/searching course codes or regNos.

- **Strings** & common methods (substring, split, join, equals, compareTo, etc.).

OOP & Type System

- **Four pillars**:
  - **Encapsulation**: private fields + **getters/setters**.
  - **Inheritance**: e.g., `Person` (abstract) → `Student` and `Instructor`.
  - **Abstraction**: `abstract class Person` with abstract methods.
  - **Polymorphism**: common interface or base-class references to varied subtypes (**virtual method invocation**).
- **Access levels**: use **private**, **protected**, **default**, **public** meaningfully.
- **Types of inheritance** and **constructors in inheritance**; demonstrate `super`.
- **Immutability**: one immutable value class (e.g., `Name` or `CourseCode`) with **final** fields and defensive copying.
- **Top-level & nested classes**: include one **static nested class** and one **inner class**.
- **Interfaces**:
  - Define at least one **interface** (e.g., `Persistable`, `Searchable<T>`).
  - Show **diamond problem** resolution via default methods and explicit override.
  - Decide where **interface vs class inheritance** makes sense (justify briefly in README).
- **Functional interfaces & lambdas**: e.g., comparator lambdas for sorting, predicates for filtering.
- **Anonymous inner classes**: use once (e.g., custom listener/callback in CLI or a quick strategy).
- **Enums** with **constructors & fields**: `Semester`, `Grade`.

Advanced Concepts

- **Upcast & downcast**, and `instanceof` checks (justify necessity in comments).
- **Overriding & overloading** methods; override `toString()` in domain classes.
- **Design patterns**:
    - **Singleton**: `AppConfig` or `DataStore`.
    - **Builder**: `Course.Builder` or `Transcript.Builder`.
- **Exceptions**:
    - Clarify **Errors vs Exceptions** in README.
    - Use checked & unchecked exceptions; **try/catch/multi-catch/finally/throw/throws**.
    - Create at least **two custom exceptions** (e.g., `DuplicateEnrollmentException`, `MaxCreditLimitExceededException`).
    - Use **assertions** for invariants (e.g., non-null ids, credit bounds) with a README note on enabling assertions.
- **File I/O (NIO.2)**:
    - **Path** & **Files** APIs for check/delete/copy/move.
    - Use **Streams** to read/write and process lines.
    - Demonstrate a small **Stream pipeline** that aggregates reports (e.g., GPA distribution).
- **Date/Time API** for timestamps (enrollment date, backup folder names).

---

4) Suggested Package Design (example)

edu.ccrm
 ├─ cli/ // Menu, input loop
 ├─ domain/ // Person (abstract), Student, Instructor, Course, Enrollment, Grade (enum), Semester (enum), immutable value objects
 ├─ service/ // StudentService, CourseService, EnrollmentService, TranscriptService (polymorphic interfaces/impls)
 ├─ io/ // ImportExportService (NIO.2), BackupService, parsers (CSV)
 ├─ util/ // Validators, Comparators (lambdas), recursion utilities
 └─ config/ // AppConfig (Singleton), builders

5) Minimum Demo Flow (what we should see when we run it)

1. On start, **AppConfig (Singleton)** loads config (e.g., data folder path).
2. CLI **switch** menu:
    - Manage Students / Courses / Enrollment / Grades
    - Import/Export Data
    - Backup & Show Backup Size (recursive)
    - Reports (e.g., top students, GPA distribution using Streams)
    - Exit
3. Perform enrollments, record grades, print **transcript** (uses polymorphism, `toString()`).
4. Export data and run **backup**. Show the generated timestamped folder.
5. Program prints a short **platform note** (from README) summarizing **Java SE vs ME vs EE**.

---

6) What to Submit (deliverables)

1. **Git repository link** with:
    - **Source code** and a runnable `main` class.
    - **README.md** containing:
        - Project overview & how to run (JDK version, commands).
        - **Evolution of Java** (short bullets).
        - **Java ME vs SE vs EE** comparison.
        - **JDK/JRE/JVM** explanation.
        - **Windows install steps** (with your screenshots) and **Eclipse** setup steps (with screenshots).
        - Mapping table: **syllabus topic → file/class/method** where it's demonstrated.
        - Notes on enabling **assertions** and sample commands.
    - **Screenshots** folder:
        - JDK installation verification (`java -version`)
        - Eclipse project setup & run
        - Program running (menus, sample operations)
        - Exports/Backups folder structure
    - **Optional demo video** link (YouTube/Drive) showing a 2–5 min walkthrough.
2. A short **USAGE.md** (or section in README) with sample commands & data files.
3. A simple **test-data** folder with small CSVs for import.

Reminder: The **expected output** is a **git repo link** with the **README file**, **screenshots**, and an **optional demo video**.

Academic Integrity

Your submission must be your own work. Cite any references used in the README's acknowledgements section. Using any kind of LLMs may result in disqualification from the project based evaluation.