# Campus Course & Records Manager (CCRM)

## 1. Project Overview

The Campus Course & Records Manager (CCRM) is a comprehensive, console-based Java application designed to streamline academic administration for an educational institute. It provides functionalities for managing students, courses, enrollments, and grades. The project is built entirely on Java SE, emphasizing Object-Oriented Programming (OOP) principles, modern Java features like the Streams and NIO.2 APIs, and fundamental design patterns.

This application serves as a practical demonstration of core Java concepts, from basic syntax and control flow to advanced topics like exception handling, file I/O, and functional programming with lambdas.

## 2. Features

- **Student Management:** Add, list, update, and deactivate student records. View detailed student profiles and academic transcripts.
- **Course Management:** Create, list, update, and deactivate courses. Assign instructors and manage course details.
- **Advanced Search:** Filter and search for courses by instructor, department, or semester using the Java Stream API.
- **Enrollment & Grading:** Enroll and unenroll students in courses, record marks, compute letter grades, and calculate GPA.
- **File Operations:** Import and export student and course data from/to simple CSV-like text files using Java NIO.2.
- **Data Backup:** A utility to create timestamped backups of all application data.
- **Recursive Utilities:** Includes a recursive function to calculate the total size of a backup directory.
- **Interactive CLI:** A user-friendly, menu-driven command-line interface for easy navigation and operation.

## 3. How to Run the Application

1. **From Eclipse IDE:**
   - Locate the main class with the main method (e.g., in the edu.ccrm.cli package).
   - Right-click on the file and select Run As > Java Application.
2. **From the Command Line:**
   - Navigate to the project's root directory.

- Compile the source code:
  ```
  javac -d bin src/edu/ccrm/cli/Main.java  // Adjust path as needed
  ```
- Run the compiled application:
  ```
  java -cp bin edu.ccrm.cli.Main
  ```

### Note on Enabling Assertions

This project uses assert statements for internal checks. To run the application with assertions enabled, use the -ea flag:

```
java -ea -cp bin edu.ccrm.cli.Main
```

## 4. Usage & Sample Commands

The application is operated through a menu-driven console. Upon starting, you will be presented with a main menu.

**Main Menu:**

1. Manage Students
2. Manage Courses
3. Manage Enrollment & Grades
4. Import/Export Data
5. Backup Data
6. Show Backup Size (Recursive)
7. View Reports
8. Exit

Simply type the number corresponding to your choice and press Enter. Follow the on-screen prompts for each sub-menu.

- **Sample Data:** The /test-data directory contains sample students.csv and courses.csv files that can be used with the "Import Data" feature.

*Example of the running application:*

## 5. Project Structure

The project is organized into logical packages to separate concerns and improve maintainability.

```
edu.ccrm
├── cli/      # Contains the command-line interface, menu, and input handling.
├── config/   # Singleton configurations and Builder pattern implementations.
```

```
├── domain/    # Core entity classes (Student, Course), enums, and value objects.
├── io/        # Handles file import/export and backup services using NIO.2.
├── service/   # Business logic for managing students, courses, and enrollments.
└── util/      # Utility classes, validators, comparators, and recursive helpers.
```

## 6. Core Java Concepts Explained

### Evolution of Java (Timeline)

- **1995:** Java 1.0 released by Sun Microsystems.
- **1998:** J2SE 1.2 (Java 2) released with Swing, Collections Framework.
- **2004:** J2SE 5.0 (Tiger) introduced Generics, Enums, Autoboxing.
- **2011:** Java SE 7 released with NIO.2, try-with-resources.
- **2014:** Java SE 8 introduced Lambdas, Streams API, Date/Time API.
- **2018:** Java SE 11 released as a Long-Term Support (LTS) version.
- **2021:** Java SE 17 released as the next LTS version.

### Java ME vs. SE vs. EE

| Feature | Java ME (Micro Edition) | Java SE (Standard Edition) | Java EE (Enterprise Edition) |
|---|---|---|---|
| **Target** | Resource-constrained devices | Desktop, server, and console applications | Large-scale, distributed enterprise applications |
| **Core API** | Subset of Java SE API | The core Java programming platform | Superset of Java SE API |
| **Includes** | CLDC, CDC (VMs for small devices) | JDK, JRE, JVM, Core Libraries (Collections, I/O) | APIs for web services (JAX-RS), servlets, JPA, etc. |
| **Use Case** | Mobile phones (legacy), embedded systems | General-purpose programming, desktop apps (Swing/FX) | Web servers, application servers, enterprise software |

### Java Architecture: JDK, JRE, JVM

- **JVM (Java Virtual Machine):** An abstract machine that provides the runtime environment in which Java bytecode can be executed. It interprets the bytecode into native machine code.
- **JRE (Java Runtime Environment):** A software package that contains what is required to run a Java program. It includes the JVM, core libraries, and other supporting files.
- **JDK (Java Development Kit):** A full-featured software development kit for Java. It contains everything in the JRE, plus development tools like the compiler (javac) and debugger (jdb).

**Relationship:** JDK > JRE > JVM. You need the JDK to develop Java applications, and the JRE to run them.

### Errors vs. Exceptions

- **Error:** Represents a serious problem that a reasonable application should not try to catch, such as OutOfMemoryError or StackOverflowError. These are issues external to the application that are typically unrecoverable.
- **Exception:** Represents a condition that a reasonable application might want to catch. There are two types:
  - **Checked Exception:** An exception that the compiler forces you to handle (e.g., IOException). They represent recoverable conditions.
  - **Unchecked (Runtime) Exception:** An exception that is not checked at compile time (e.g., NullPointerException, IllegalArgumentException). They often indicate programming errors.

## 7. Mapping of Syllabus Topics to Code

| Syllabus Topic | File / Class / Method Where Demonstrated |
|---|---|
| **OOP: Encapsulation** | Student.java, Course.java (private fields with public getters/setters) |
| **OOP: Inheritance** | Person.java (abstract base), Student.java, Instructor.java (subclasses) |
| **OOP: Abstraction** | Person.java (abstract class with abstract methods) |
| **OOP: Polymorphism** | TranscriptService.java (using Person references for Student objects) |

| | |
|---|---|
| **Interfaces & Lambdas** | Searchable<T> interface, used with lambda predicates in CourseService |
| **Design Pattern: Singleton** | AppConfig.java (ensures a single instance for application configuration) |
| **Design Pattern: Builder** | Course.java (contains a static nested CourseBuilder class) |
| **Exception Handling (Custom)** | DuplicateEnrollmentException.java, MaxCreditLimitExceededException.java |
| **File I/O (NIO.2)** | ImportExportService.java (uses Path, Files, and Streams) |
| **Java Stream API** | CourseService.java (searchCourses method uses filter and collect) |
| **Java Date/Time API** | Enrollment.java (uses LocalDate for enrollment dates) |
| **Recursion** | BackupService.java (getDirectorySizeRecursive method) |
| **Enums** | Grade.java, Semester.java (with constructors and fields) |

## 8. Acknowledgements

- This project was completed as a part of the "Programming in Java" course.
- Documentation from Oracle on Java SE was used as a reference.

## 9. License