Campus Course & Records Manager (CCRM)

Author: Akshat Shukla

Abstract

The **Campus Course & Records Manager (CCRM)** is a Java SE console application for managing student data, courses, enrollment, grading, transcripts, and utilities such as import/export and backup. It is designed to illustrate key Java programming paradigms (OOP, design patterns, streams, exception handling, recursion, and file I/O) in the context of an academic records system. The system's architecture is modular (with layers for CLI, domain, services, IO, etc.), and the implementation leverages modern Java features (Streams API, NIO.2 file API, Date/Time API, enums). This paper presents the motivation for the project, surveys related systems, details the design and implementation, and discusses results such as CLI-driven workflows (student/courses management, GPA calculation, backup) and limitations. Finally, future extensions (e.g. GUI, database integration, REST API) are proposed to enhance CCRM's functionality and realism.

1. Introduction

Managing academic records – including student profiles, course catalogs, enrollments, grades, and transcripts – is a core administrative function in universities. Large institutions often rely on comprehensive student information systems (SIS) or Enterprise Resource Planning (ERP) software. For example, **OSIS (Open Student Information System)** is "an application designed to manage the core business of higher education institutions" such as universities and colleges[1], covering students, teachers, courses, programs, etc. Similarly, **ERPNext** offers an education module that is "a comprehensive solution to manage admissions, student records, fees, and learning outcomes"[2], including course scheduling, grade recording, and transcript generation[3]. However, such systems are often complex, web-based, and heavyweight – whereas there is value in lightweight educational tools and prototypes to teach programming concepts or serve small-scale needs.

This work presents CCRM, a **Java SE** (console-based) application that provides core functionalities of a campus records system while highlighting modern Java practices. CCRM supports adding/removing students and courses, enrolling students, entering grades, computing GPA and transcripts, and performing data import/export and backups. Its design emphasizes clear modular structure and patterns: for example, configuration and data store use the Singleton pattern; `Course` objects are built via a Builder; domain classes (`Student`, `Instructor`, etc.) use inheritance and enums for controlled values. The implementation uses Java 8+ features (Streams API for collection processing, `java.time` for dates, NIO.2 for file I/O) and robust exception handling. In sum, CCRM is a minimal but extensible framework for campus-level record keeping, suitable for teaching or prototype purposes.

2. Literature Review

2.1 Evolution of Java

Java SE has evolved since its 1995 inception, adding key features each release. **Java 5 (2004)** introduced major language enhancements: generics (type parameters for collections), annotations (metadata), and enums (typesafe constant sets)[4]. Java 7 (2011) added the **NIO.2** file I/O API (`java.nio.file`) for more robust and flexible file operations (including `Path` and `Files` classes)[5]. **Java 8 (2014)** brought lambda expressions and the **Streams API**, enabling functional-style processing of collections. The Stream API greatly simplifies data manipulation: as Baeldung notes, streams "allow us to reduce a huge amount of boilerplate code [and] create more readable programs"[6]. Subsequent releases (Java 9–17) added refinements, but the core features used in CCRM (streams, NIO.2, `java.time`, etc.) were mature by Java 8–11. CCRM is built on **Java SE 17** (the current long-term support release), benefiting from all these features (for example, the modern `java.time` API for dates introduced in Java 8).

2.2 State-of-the-Art Educational Systems

A variety of educational record systems exist today, typically as web or enterprise software. **RosarioSIS** is a free open-source *Student Information System* (SIS) originally for K–12 schools. It "features students demographics, grades, scheduling, attendance, student billing, discipline & food service modules" in one web application[7]. Similarly, **ERPNext** (a full ERP suite) offers an Education module where educators can "organize courses, schedule classes, and track learning progress" while teachers "record grades, manage assignments, and generate transcripts effortlessly"[3]. These systems are usually backed by databases, GUIs, web integration, and broad feature sets (e.g. fees, attendance, learning management) that go beyond core record keeping.

While such systems are powerful, they can be too complex for teaching purposes or small schools. Many academic programming courses instead use simplified console-based projects (e.g. "Student Management System in Java") focusing on basic CRUD operations. CCRM positions itself between these extremes: it remains a **console (CLI) application** for simplicity, but incorporates advanced Java features and realistic workflows (like transcript generation and backups) often missing in simple tutorials. In summary, current practice ranges from large, full-featured ERP/SIS platforms[1][2][7] to minimal academic examples; CCRM aims to blend completeness with educational clarity.

3. System Design

3.1 Architecture

CCRM follows a layered modular architecture to separate concerns. The main packages under `edu.ccrm` are organized as follows:

```
src/edu/ccrm/
├─ cli/      (MainMenu and CLI user interface code)
├─ config/   (Singleton classes: AppConfig for settings, DataStore for in-
```

```
memory storage)
├─ domain/   (Core entities: Student, Course, Instructor, Enrollment, Grade,
Semester, etc.)
├─ service/  (Business logic: StudentService, CourseService,
EnrollmentService, etc.)
├─ io/       (Data utilities: CSV import/export, BackupService using NIO.2)
├─ util/     (Helper classes: Validators, date/recursion utilities)
└─ exception/ (Custom exception classes for domain errors)
```

The **`cli`** package drives all user interactions via a text menu. The **`config`** layer includes single-instance classes: `AppConfig` (application settings) and `DataStore` (in-memory collections of students, courses, etc.). The **`domain`** package contains the core data classes with appropriate fields (for example, `Course` has code, title, credits, etc.; `Student` has name, DOB, ID). The **`service`** layer implements operations on the domain (adding students/courses, enrolling, calculating GPA, etc.). The **`io`** layer handles data import/export and backups: for instance, an `ImportService` reads CSV files into objects, and `BackupService` uses NIO.2 to copy files with timestamped directories. Utility classes and custom exceptions support validation and error handling across layers.

3.2 Design Patterns and Key Structures

CCRM uses well-known patterns and language features for robustness and clarity:

- **Singleton:** `AppConfig` and `DataStore` are implemented as singletons, ensuring a single global configuration and data repository. This simplifies access to shared data without passing objects everywhere.

- **Builder:** `Course` is designed as an immutable class constructed via an inner `Course.Builder`. The *Builder Pattern* "separates the construction of complex objects from their representation"[8]. Using a builder for `Course` allows step-by-step setting of attributes (like code, title, credits) and then calling `build()`, yielding a thread-safe, final `Course` object. This avoids telescoping constructors and enforces immutability (beneficial in multi-threaded contexts)[9].

- **Inheritance/Polymorphism:** A base class `Person` defines common fields (name, DOB) and a method `profileSummary()`. `Student` and `Instructor` extend `Person` and override `profileSummary()` to provide role-specific details. This polymorphic behavior allows treating all people uniformly in some contexts while still differentiating students vs. instructors.

- **Enum Types:** Java `enum` classes (introduced in Java 5[4]) represent fixed academic values. For example, `Semester` is an enum (e.g. FALL, SPRING), and `Grade` is an enum (A, B, C, etc.). Using enums enforces type safety and easy comparisons (instead of raw strings or ints). This prevents invalid values for grades or semester identifiers.

- **Collections and Streams:** The implementation uses Java collections (e.g. `List<Student>`). Data filtering and aggregation (e.g. finding a student by ID, computing

GPA) often use the Streams API, which "has become a staple of Java development" due to its conciseness[6] (see Implementation below).

The following ASCII diagram illustrates the package layout (some details omitted for brevity):

```
edu.ccrm
├─ cli/           # MainMenu CLI interface
├─ config/        # Singleton AppConfig, DataStore
├─ domain/        # Core entities (Student, Course, etc.)
├─ service/       # Business logic (e.g., enrollment, grading)
├─ io/            # CSV import/export, backup utilities
├─ util/          # Misc utilities (validators, recursion, etc.)
└─ exception/     # Custom exception classes (e.g.,
DuplicateEnrollmentException)
```

4. Implementation

4.1 Technologies

CCRM is built with **Java SE 17** using standard libraries and a typical IDE (Eclipse or IntelliJ). Key technologies include:

- **Java SE 17:** The latest long-term support version, supporting all modern language features used.
- **Streams API:** Java's Streams are heavily used for data processing. For example, filtering a list of courses by department or summing grades with `.mapToDouble(...).average()`. As noted, Streams reduce boilerplate when processing collections[6].
- **NIO.2 (`java.nio.file`):** This package provides enhanced file I/O (introduced in Java 7). It is used for import/export of CSV and for the backup system. The official tutorial states that `java.nio.file` "provides comprehensive support for file I/O" and is "very intuitive and easy to use"[10]. Classes like `Path` and `Files` allow robust operations (copying, deleting, iterating directories).
- **Date/Time API (`java.time`):** Used for student DOB and record timestamps. This modern API (JSR-310) handles dates cleanly and immutably, replacing older `Date` classes.
- **Collections:** Standard Java collections (`List`, `Map`, etc.) store in-memory data. CCRM does not use a database in this version; all data is lost when the program exits (data persistence is future work).
- **Others:** CCRM uses a simple console `Scanner` for CLI input, and writing to standard output. No external libraries (beyond Java SE) are required.

4.2 CLI Interaction

The user interacts with CCRM exclusively via a text menu. On startup, a **Main Menu** is displayed with numbered options. For example:

1. **Manage Students** – Add, view, delete student records.
2. **Manage Courses** – Add, view, delete course entries.
3. **Enrollment & Grades** – Enroll a student in a course or assign grades.
4. **Import/Export** – Load or save data in CSV format.
5. **Backup & Reports** – Create backups of data and generate reports (transcripts, GPA lists).

Each option leads to sub-menus or actions; the interface loops until the user chooses to exit. A sample transcript report for a student might appear as:

```
Transcript for Akshat Shukla
ENR-1000 | Akshat Shukla -> C101 | marks=87.00 grade=A
GPA: 9.00
```

This shows the student's name, enrollments (courses with marks and letter grade), and computed GPA. The CLI is kept text-based to focus on backend logic; designing a GUI or web UI is deferred to future work.

4.3 Exception Handling

To ensure robustness, CCRM defines custom exception classes for domain-specific errors. For example:
- `DuplicateEnrollmentException` is thrown if a student is enrolled twice in the same course.
- `MaxCreditLimitExceededException` is used if enrolling a student would exceed a predefined credit limit.

Using **custom exceptions** is considered good practice when built-in exceptions are insufficient. As one source notes, custom exceptions allow adding application-specific information (e.g. an error code) and make APIs clearer[11]. In CCRM, catching these exceptions in the CLI layer enables user-friendly error messages (e.g. "Error: Student is already enrolled in that course.") without exposing internal logic. The pattern of throwing and catching domain-specific exceptions improves code clarity and maintainability.

4.4 Recursion (Backup)

The backup subsystem uses **recursive directory traversal** to process files. For instance, `BackupService` may need to compute the total size of a folder before zipping it. CCRM implements this with a recursive helper that visits each file and subdirectory. This mirrors the approach shown in the Java tutorials: the official NIO.2 guide even demonstrates "how to recursively walk the file tree" using APIs like `Files.walkFileTree`[12]. By analogy, CCRM's recursive method descends into directories via `File.listFiles()` (or `Files.newDirectoryStream`) and sums file sizes, demonstrating practical recursion for file I/O.

5. Results & Discussion

CCRM effectively demonstrates core object-oriented and modern Java practices within an academic record system context. Key observations include:

- **OOP & Encapsulation:** Entities like `Student` and `Course` encapsulate data and related behaviors. Inheritance (via a `Person` base class) reduces code duplication, and `profileSummary()` polymorphism allows uniform handling of different roles.

- **Academic Workflows:** The main use cases (adding students/courses, managing enrollments, grading, and generating transcripts) are fully supported. The GPA calculation, for example, converts numeric marks to grade points and averages them. These workflows simulate real registrar functions at a campus level.

- **Data Utilities:** CCRM can import student and course lists from CSV files (converting lines to objects) and export current records back to CSV. It also creates timestamped backups of its data folder. These features show how Java's NIO.2 and I/O APIs can handle common data tasks (file reading, writing, copying).

- **Robustness:** Input validation and custom exceptions prevent invalid operations (e.g. enrolling non-existent students). The code handles errors gracefully, printing helpful messages rather than crashing. This robustness is crucial even in a console app.

- **Performance:** With in-memory storage, CCRM can handle on the order of thousands of students and courses easily on a typical computer. The main limitations are console I/O speed and CPU for processing streams over lists, but these are not problematic for this scale.

Limitations: CCRM's simplicity has trade-offs. It lacks a graphical or web interface (user must use CLI), and all data is lost on exit (no database persistence). There is also no authentication or multi-user support. In practice, a production SIS would use a database (MySQL, PostgreSQL, etc.) and a user-friendly UI. Despite these, CCRM achieves its educational goals by clearly illustrating Java features and design.

6. Conclusion & Future Work

The Campus Course & Records Manager (CCRM) provides a practical demonstration of applying modern Java SE features to a student records domain. It strikes a balance between **educational clarity** (simple CLI, modular code) and **realistic functionality** (courses, enrollment, grading, transcripts). The system successfully showcases object-oriented design (OOP principles and patterns), Java APIs (Streams, NIO.2, time), and good coding practices (exception handling, immutability).

Future enhancements would bring CCRM closer to a full-scale system: adding a GUI or web front-end for usability; integrating a relational database for persistent storage; exposing RESTful APIs for interoperability; and extending domain logic (e.g. attendance tracking, fee management, analytics dashboards). These would require additional technologies (JDBC, web frameworks like Spring, etc.) but would build on the solid foundation established here. In summary, CCRM is a starting point for a campus management tool and a case study in applying Java SE techniques to an academic software problem.

7. References

- Oracle Java Tutorials: *File I/O (Featuring NIO.2)* – Comprehensive guide to `java.nio.file` and file operations[10][12].
- Baeldung: *The Java Stream API Tutorial* (Oct 2023) – Overview of Java Streams, highlighting reduced boilerplate and improved readability[6].
- Baeldung: *Java – Path vs File* – Explanation of Java 7 NIO.2 `Path` class (part of the Path API introduced in Java 7)[5].
- Baeldung: *Implement the Builder Pattern in Java* – Description of the Builder design pattern and its advantages in object construction[8][9].
- Stackify: *Why, When and How to Implement Custom Exceptions in Java* (2017) – Discussion of best practices for creating custom exceptions, including adding application-specific context[11].
- OSIS (Open Student Information System) – Project site describing an open-source SIS for higher education (managing students, courses, etc.)[1].
- ERPNext for Education (Frappe) – Documentation and marketing describing ERPNext modules for schools (admissions, student records, fees, grades, transcripts)[2][3].
- RosarioSIS – Official site for an open-source student information system (K-12 focused) with modules for demographics, grades, scheduling, attendance, billing, etc.[7].
- Oracle Java SE 5.0 Release Notes – Lists new language features in Java 5 (generics, annotations, enums)[4].

---

[1] OSIS - Open Student Information System

https://uclouvain.github.io/osis/

[2] [3] Open Source ERP Software for Educational Institutions | ERPNext

https://frappe.io/erpnext/for-education

[4] J2SE(TM) 5.0 New Features

https://docs.oracle.com/javase/1.5.0/docs/relnotes/features.html

[5] Java – Path vs File | Baeldung

https://www.baeldung.com/java-path-vs-file

[6] The Java Stream API Tutorial | Baeldung

https://www.baeldung.com/java-8-streams

[7] RosarioSIS | Free Student Information System for school management

https://www.rosariosis.org/

[8] [9] Implement the Builder Pattern in Java | Baeldung

https://www.baeldung.com/java-builder-pattern

[10] [12] File I/O (Featuring NIO.2) (The Java™ Tutorials > Essential Java Classes > Basic I/O)

https://docs.oracle.com/javase/tutorial/essential/io/fileio.html

[11] Implement Custom Exceptions in Java: Why, When and How

https://stackify.com/java-custom-exceptions/