**1. Class Structure & Variables**

**Class:** `StudentRecordHub`

**Purpose:** Central class handling all student record operations through static methods and variables

**Variables:**

static final int MAX\_CAPACITY = 100; // Maximum student capacity

static String[] studentNames = new String[MAX\_CAPACITY]; // Stores student names

static String[] studentIDs = new String[MAX\_CAPACITY]; // Stores unique student IDs

static int[] studentAges = new int[MAX\_CAPACITY]; // Stores student ages

static String[] studentGrades = new String[MAX\_CAPACITY]; // Stores letter grades

static int studentCount = 0; // Tracks current student count

**2. Core Operations**

**A. Adding New Students**

**Method:**`enrollNewStudent()`

**Process Flow:**

1. Checks system capacity

2. Validates inputs through:

- `validateName()`: Ensures non-empty names

- `generateUniqueID()`: Prevents duplicate IDs

- `validateAge()`: Restricts ages 12-25

- `validateGrade()`: Accepts only valid letter grades

**3. Stores validated data via `storeStudentData()`**

**Code Example:**

private static void enrollNewStudent(Scanner scanner) {

if (studentCount >= MAX\_CAPACITY) {

System.out.println("\n⚠️ System at full capacity");

return;

}

// Input validation sequence

String name = validateName(scanner);

String id = generateUniqueID(scanner);

int age = validateAge(scanner);

String grade = validateGrade(scanner);

// Data storage

studentNames[studentCount] = name;

studentIDs[studentCount] = id;

studentAges[studentCount] = age;

studentGrades[studentCount] = grade;

studentCount++;

}

**B. Updating Student Information**

**Method: `modifyStudentRecord()`**

**Features:**

* ID-based student search using `findStudentIndex()`
* Partial updates preserving existing values
* Real-time validation during editing

**Update Process:**

private static void updateStudentDetails(Scanner scanner, int index) {

// Name update

System.out.print("New Name [current: " + studentNames[index] + "]: ");

String newName = scanner.nextLine();

if (!newName.isBlank()) studentNames[index] = newName;

// Age update with validation

System.out.print("New Age [current: " + studentAges[index] + "]: ");

String ageInput = scanner.nextLine();

if (!ageInput.isBlank()) {

try {

int newAge = Integer.parseInt(ageInput);

if (newAge >= 12 && newAge <= 25) {

studentAges[index] = newAge;

}

} catch (NumberFormatException e) {

System.out.println("⚠️ Invalid age format");

}

}

}

C. Viewing Student Details

Method: `displayStudentProfile()`

Functionality:

1. Accepts student ID input

2. Searches records via `findStudentIndex()`

3. Displays formatted profile

Output Example:

📄 Student Profile

Name : Sarah Johnson

ID : SJ-2024

Age : 18

Grade : A

**3. Error Handling System**

**Validation Mechanisms:**

**1. Age Validation**

private static int validateAge(Scanner scanner) {

while (true) {

try {

int age = Integer.parseInt(scanner.nextLine());

if (age >= 12 && age <= 25) return age;

System.out.println("❌ Age must be 12-25");

} catch (NumberFormatException e) {

System.out.println("❌ Numbers only please");

}

}

}

**2. Grade Validation**

private static boolean isValidGrade(String grade) {

return grade.matches("A|B\\+?|C\\+?|D|F");

}

**3. ID Uniqueness Check**

private static String generateUniqueID(Scanner scanner) {

while (true) {

String id = scanner.nextLine().toUpperCase();

if (findStudentIndex(id) == -1) return id;

System.out.println("❌ ID exists. Try again");

}

}

**4. User Guide**

**Running the Program:**

1. Compile: `javac StudentRecordHub.java`

2. Run: `java StudentRecordHub`

**Menu Navigation:**

========= Management Console =========

1. Enroll New Student

2. Modify Existing Record

3. View Student Profile

4. Exit System

**Input Requirements:**

* Names: Non-empty text
* Ages: Numbers 12-25
* Grades: A, B+, B, C+, C, D, F

**5. Technical Specifications**

**Search Algorithm:**

private static int findStudentIndex(String id) {

String targetID = id.toUpperCase();

for (int i = 0; i < studentCount; i++) {

if (studentIDs[i].equals(targetID)) return i;

}

return -1;

}

**Complexity: Linear Search (O(n))**

**Data Storage:**

* Parallel array structure
* Automatic case conversion (IDs/grades)
* Atomic transaction handling

**6. Limitations & Future Improvements**

**Current Constraints:**

* Maximum 100 student records
* No data persistence between sessions
* Single-user access

**Proposed Enhancements:**

1. File-based data storage

2. Database integration

3. Multi-field search capability

4. Bulk import/export