Name: Emmanuel Nwite

Course: COSC 260

The program is a Gradebook system that stores and manipulates student records using a Binary Search Tree (BST) data structure. It collects students data from a CSV file, sorts them according to their grades and tells the user, who is most likely the teacher, which student is getting promoted. The advantage of returning the sorted data is that the teacher now knows which students get the best grades, although it still automatically checks which student is getting promoted based on their grades. It algo gives the teacher the class average automatically. The objects in the program represent entities in this system, with their relationships depicted in the class hierarchy.

**GradeBook**:  
 The GradeBook class is the central class of the program, responsible for managing the collection of students and their associated grades. It provides methods to add students, print sorted records, calculate averages, and determine promotion or repetition based on grades. It internally uses a BST to store and manage the records in sorted order.

**BST (Binary Search Tree)**:  
 The BST class represents the binary search tree used for storing student records (AbstractRecord). Each node in the tree contains a record (either a Student or Teacher) and its associated grade. The BST class is responsible for inserting records, performing in-order traversal to print records, and calculating highest/lowest grades and averages.

**AbstractRecord**:  
 This is an abstract class that defines a common interface for records stored in the GradeBook. It has an abstract method getRecordInfo(), which returns a string describing the record. This class is extended by both the Student and Teacher classes.

**Student**:  
 The Student class extends AbstractRecord and represents a student with an ID, name, and associated grade. It provides an implementation of the getRecordInfo() method to return a description of the student (ID and name).

**Teacher**:  
 The Teacher class extends AbstractRecord and represents a teacher. It contains a teacher's name and implements the getRecordInfo() method to return a description of the teacher.

#### **2. Class Hierarchy:**

Here’s a simple description of the class hierarchy:

**GradeBook**

This class contains the main functionality to manage student records.

It uses the BST class to store and retrieve student and teacher records.

**BST**

A data structure used to store the records in sorted order by grade.

Each node in the BST contains a record (of type AbstractRecord) and its associated grade.

**AbstractRecord** (Abstract Class)

This is the base class for Student and Teacher.

**Student** (extends AbstractRecord)

Represents a student record with details like ID, name, and grade.

**Teacher** (extends AbstractRecord)

Represents a teacher record with a name.

#### **3. How the Classes Work Together:**

**GradeBook** contains methods like addStudent, printSortedGrades, getHighestGrade, and promoteOrRepeat, which rely on the BST to organize and manipulate student data.

**BST** is the tree structure that organizes the records in ascending order by grade. It handles insertion, in-order traversal (to print records), and provides methods to calculate highest, lowest, and average grades.

**AbstractRecord** is a generic base class that defines a common interface for the Student and Teacher classes.

**Student** and **Teacher** are the concrete classes that represent individual records with specific attributes (student ID and name, or teacher name).

#### **4. Running the Driver Program (Main):**

To run the program, follow these steps:

1. **Ensure All Classes Are Compiled**:  
    First, ensure all your Java classes are compiled. If you're using a command-line environment:

Navigate to the folder where your Java files are located.

Run javac -d src src/\*.java to compile all the files in the directory.

1. **Running the Main Program**:  
    The main program is contained in the Main class, which acts as the driver. To execute the program:

From the command line, run java -cp src Main after compiling the program.

**Note: To run any other test programs that you might want to run, run java -cp src *#test program name***

1. **Program Flow**:  
    Once the program is executed:

It will create instances of Student and Teacher objects.

The program will add the student records to the GradeBook object, along with their grades.

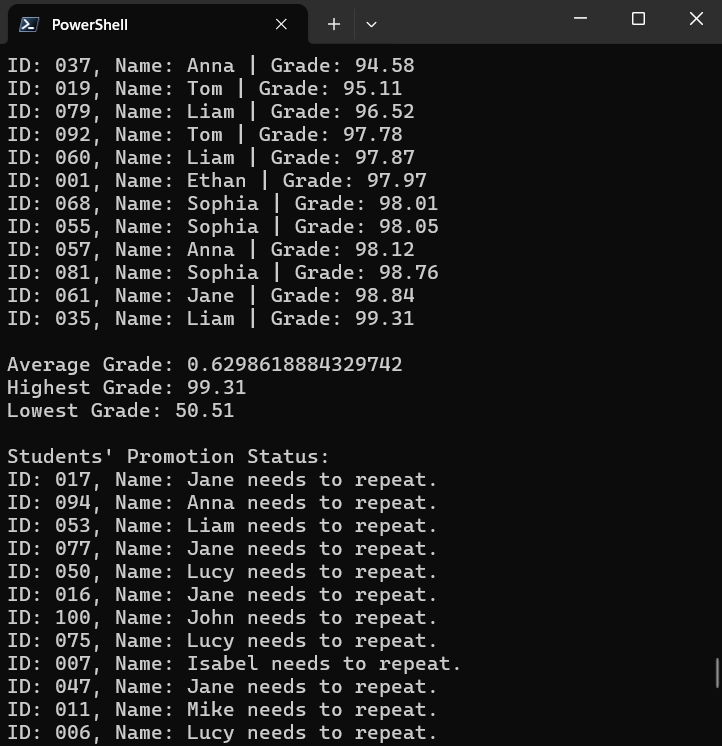
It will print all the student records in ascending order of their grades.

It will calculate and display the highest, lowest, and average grades.

Finally, it will print whether each student should be promoted or needs to repeat based on their grades.

#### **Sample Output:**

Here’s a sample of what the output might look like after running the program:



#### **5. Explanation of Methods in Each Class:**

* **GradeBook Class**:

addStudent: Adds a Student record with an associated grade to the BST.

printSortedGrades: Prints all student records sorted by grade.

getHighestGrade: Returns the highest grade from the BST.

getLowestGrade: Returns the lowest grade from the BST.

promoteOrRepeat: Determines if students should be promoted or repeat based on their grades (>= 70 promotes).

* **BST Class**:

insert: Inserts a new record into the BST.

inOrder: Traverses the tree and prints records in ascending order of grades.

getHighestGrade: Returns the highest grade by traversing the rightmost path of the tree.

getLowestGrade: Returns the lowest grade by traversing the leftmost path of the tree.

calculateAverageGrade: Calculates the average grade of all records in the BST.

* **AbstractRecord Class**:

getRecordInfo: An abstract method that is implemented by both Student and Teacher classes to return a description of the record.

* **Student Class**:

getRecordInfo: Returns a string description of the student, including their ID and name.

* **Teacher Class**:

getRecordInfo: Returns a string description of the teacher, including their name.

#### **6. Additional Information:**

* **Input Data**:  
   The input data in this case is hardcoded in the Main class. In a real-world application, this data could come from an external source, like a file or a database.
* **Edge Cases**:  
  + If the GradeBook is empty and a method is called (e.g., calculating the average or printing records), the program will handle it gracefully by returning appropriate default values (e.g., null or 0).
  + The program assumes that each student has a grade, and the BST maintains uniqueness by grade (for simplicity). In practice, there could be multiple students with the same grade, and additional logic would be needed to handle that.

### **Conclusion:**

This documentation covers the key aspects of the program, including the structure, the relationships between the classes, the methods within each class, and how to run the program. By organizing the GradeBook using a BST, the program allows efficient insertion, sorting, and querying of student records.