PBL: Grade Book Part 8: `class AssignmentLoader`

This assignment continues the project to create software for a school grade book that will hold the scores that students received for their assignments.

This part of the assignment will implement `class AssignmentLoader` that will load a file that contains both the information about an assignment to create an `Assignment` object, and the score each student has received on the assignment.

The format of the file is as follows:

- The first line of text contains the assignment name.
- The second line of text contains the maximum score for the assignment as an integer, then optionally white space followed by the word "`strict`" if the student will not be able to receive a score higher than the maximum score.
- Each subsequent line is to contain the student's student number and score, separated by a comma. The remainder of the line can contain any data, such as the student's name or a comment about the assignment.

`Scores_a1.txt`

```
A1 - Primitive Types 1
10 strict
1234,10
2345,5,Huang,James J. – James was feeling ill this day.
3456,10 Long,Chen
4567,10,Zhuge,Liang
```

Implement class AssignmentLoader according to the following specification.

- The class is to contain at least these three fields:
  - A field of type `String` to store the filename where the assignment information has been saved.
  - A field of type `StudentList` to store the list of students – this is needed so that each score can be added to the appropriate student's existing assignment scores.
  - A field of type `Assignment` that will store the `Assignment` object that will be created using the details about the assignment as loaded from the file.
- A constructor that takes two parameters:
  - the filename to save in the filename field.
  - the list of students to store in the `StudentList` field.
- A load method that will load all the data from the file specified by the filename field. It must:
  - Create an assignment object and save that in the `Assignment` field.
  - Load all student scores, and for every score successfully loaded, save that score into the `Student` object it belongs to. *Hint*: the student number will be used to determine which `Student` object the score should be added to. You may wish to refer to the structural diagram in the previous part of the assignment to aid in understanding how `AssignmentScore` objects store the scores for each student. You are advised to decompose this method into simpler parts and implement one or more `private` helper methods to keep the code more understandable.
- A getter method for the `Assignment` field.

You may write a stand-alone `TestAssignmentLoader` class, or you can proceed to the next part of the assignment where all previous parts of the assignment will be brought together to make a final application. (Or rather: a simple, limited-functionality application that demonstrates how these current classes would be tied together if we were to continue to build a proper application).

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class AssignmentLoader {

    private String filename;
    private StudentList studentList;
    private Assignment assignment;

    public AssignmentLoader(StudentList studentList,
                            String filename) {
        this.filename = filename;
        this.studentList = studentList;
        this.assignment = null;
    }

    public Assignment load() {

        Scanner scanner = null;

        // The try block will catch errors when parsing file

        try {
            scanner = new Scanner(new File(filename));

            getAssignment(scanner);

            if(assignment == null) return null;

            scanner.useDelimiter("[\\s,\\n]");
            while (getNextScore(scanner))
                ;
        } catch (FileNotFoundException e) {
            System.out.println("File not found!");
        } catch (Exception e) {
            System.out.println(
                "An error occurred while reading the file: " +
                e.getMessage());
        } finally {
            if (scanner != null) {
                scanner.close();
            }
        }

        return assignment;
    }
```

```
private void getAssignment(Scanner s) {
    // Expected first line is the assignment name,
    // second line is the maximum score (an integer)
    boolean isStrict = false;
    String name = s.nextLine();
    if(!s.hasNextInt()) {
        System.out.println(
            "ERROR: Assignment maxScore not found on " +
            "line 2 of file: " + this.filename);
        return;
    }
    int maxScore = s.nextInt();
    String strictString = s.nextLine();
    if(strictString.equals("strict")) {
        isStrict = true;
    }
    this.assignment = new Assignment(name, maxScore, isStrict);
}
```

```
private boolean getNextScore(Scanner s) {
    int studentNumber;
    double score;
    if (!s.hasNext()) {
        // No more tokens, end of file
        return false;
    }
    if(!s.hasNextInt()) {
        System.out.println(
            "Error parsing student number in file: " +
            this.filename);
        return false;
    }
    studentNumber = s.nextInt();
    if(!s.hasNextDouble()) {
        System.out.println(
            "Error parsing student score in file: " +
            this.filename);
        return false;
    }
    score = s.nextDouble();
    s.nextLine();
    return giveStudentScore(studentNumber, score);
}
```

```
    private boolean giveStudentScore(int studentNumber,
                                     double score) {
        Student student = studentList.getStudent(studentNumber);
        if(student==null) {
            System.out.print("Score for unknown student number: " +
                             studentNumber);
            return false;
        }
        student.addAssignmentScore(this.assignment, score);
        return true;
    }
}
```