ExerciseSheet06

In your last exercise, you will implement a file reader and writer. More specifically, you enable the artist manager to read and write Tracks to a common .csv file.

As some exceptions could occur when working with files, you will inform your user (the manager) in a predefined way.

> Be aware that it may happen that you may have questions during implementation. Don't hesitate to ask the "managers" (your lecturers and colleagues) by posting your question to the forum!

Tasks

- Implement the abstract generic class `MyReader<T>` which is the blueprint for all your future Reader implementations
- Implement the class `MyTrackCSVReader` which is derived from `MyReader<T>`, which enables the manager to read a CSV File containing Tracks
- If you didn't before, implement the class `CSVTrackFormatter` which formats a Track in CSV style (check javadoc for more information)
- Implement a generic class `MyWriter` which allows the manager to write any object to a file by using a predefined formatter
- Implement a class `DemoApp` with a main method. In the main method, make use of the reader to read the provided CSV file. You may want to print each Track from the file to the console. Also try to cause the ecxeptions you should throw and/or catch (e.g. change a number to a string in the given csv). Do the same with the Writer- create some Tracks and write them to a CSV file by using the `CSVTrackFormatter` from the exercise before. In best case you add a text menu for ease-of-use. (recommended)

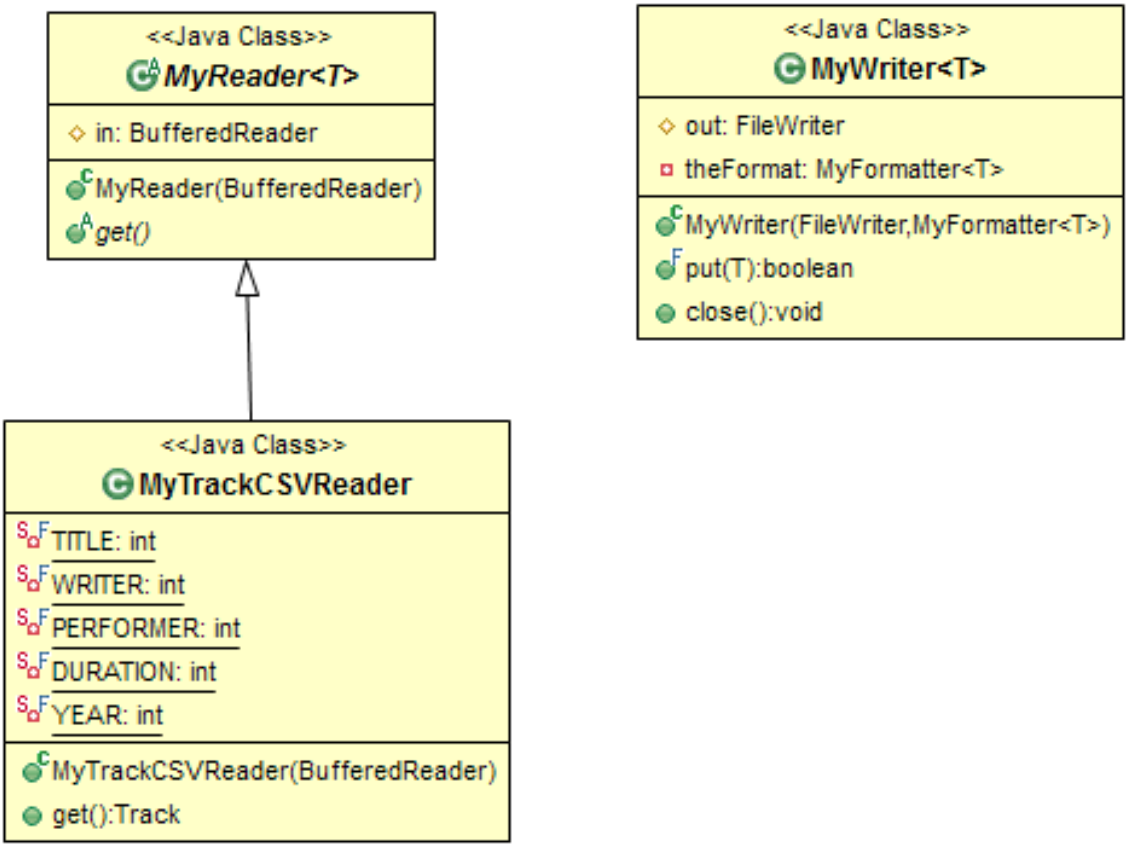

Figure 1: Classdiagram ES06 (only new/ to be changed classes)

For a detailed description of the fields and methods, please refer to the respective javadoc. Remember: You find new Methods introduced in the actual ExerciseSheet by checking the *Introduced in:* Tag.

Hint

Again, we provide some implemented code as a starting point and support.

The basic struct of `MyReader<T>`is given. Make sure you implement the constructor (isn't done yet).

```
import java.io.BufferedReader;

/**
 * Abstract class for creating Objects from data stored in files (or more
 * specifically a BufferedReaders).
 * The only method that subclasses must implement is T get() which returns an
 * object representing the next record in the underlying stream.
 * Subclasses are used to read or load data stored in textfiles (like *.csv or
 * similiar).
 *
 *
 *
 * @author TeM, JS
 *
 * @param <T>
 *            the type of the objects that can be loaded
 *
 *
 * @ProgrammingProblem.Category generic abstract class
 * @ProgrammingProblem.Category importing data from file
 *
 * @ProgrammingProblem.Introduced ExerciseSheet06
 *
 */
public abstract class MyReader<T> {

    /**
     * the underlying stream from which data is read
     */
    protected BufferedReader in;

    /**
     * constructs a MyReader from a Buffered Reader.
     * the underlying stream cannot be null. In case a null object is passed to
     * this constructor an IllegalArgumentException including
     * a custom message "expected non-null ReaderObject" is thrown.
     *
     * @param in
     *            the underlying stream
     *
     *
     * @ProgrammingProblem.Aspect throwing standard exceptions
     */
    public MyReader(BufferedReader in) {
        //TODO by yourself!
    }
```

```
    /**
     *
     * Gets the next object from the underlying stream.
     *
     * Reads the next record and creates an object with the respective values
     * set. This method handles ALL IOExceptions that might occur and returns
     * null objects in such situations.
     *
     * @return the next record as an object with all values set
     */
    public abstract T get();
}
```

Information in javadoc

You are asked to implement the Constructor in the abstract generic class `MyReader` as well as the `MyTrackCSVReader` and `MyWriter` class

For some methods/fields/classes, you may find a headline **Hint** which gives you implementation hints.

##Tests## The following amount of tested methods should be shown in your testng overview:

Classes	Methods
MyTrackCSVReader	5
MyWriter	4
sum	**9**

For testing the Reader, make sure you copied the provided mytracks.csv to the root folder of your project.

##ATTENTION##

- Make sure you have implemented all methods and fields with the predefined names!
- If you are not able to implement some of them because of the requirements, generate at least the methods themselves, or else the tests won't compile and therefore the ExerciseSheet won't be graded.
- Never change the testfiles themselves (except the neccessary changes for package name and imports), adjustments in testfiles lead to 0 points of the exercise (as grading will be done with the original ones).
- Please note that 100% passed tests don't mean 100% correct implementation