# Quote Loader Component

A junior developer was asked to create a reusable stock quote loader component, which should import quotes from a specified tab-separated file into the internal application database. Access to the database was provided by the *IQuoteRepository* interface located in the *Quotes* assembly. As a result, the junior developer produced *QuoteLoader* and *QuoteLoader.Tests* projects attached to this task.

During regular code review, a team lead pointed to a low quality of *QuoteLoader* code. One his concern was about the CSV reader/writer component, which is not generic enough to handle **various possible sources** of stock quotes. Another his comment was about **poor unit test coverage**.

## Task

Your task will be to focus on issues, which team leader considered the most critical for now, such as the CSV reader/writer component.

1. Review the existing CSV reader/writer implementation and **create a complete list of problems** you see in the code (i.e. for junior developer education purposes).
2. **Refactor** the CSV reader/writer component into a clean, performant and elegant code, without over-engineering. Make sure you follow **best practices for design** and implementation. Annotate your code with comments and explain any trade-offs you make.
3. Assume the CSV reader/writer already used in production, and **backward compatibility** must be maintained. Therefore, make sure your changes do not break any contracts. If **you decide to change** public interface in any way, please deprecate existing methods.
4. Make sure you write enough unit tests to achieve near 100% code coverage for CSV reader/writer component, also testing various exceptional cases. Feel free to introduce any mocking frameworks available in NuGet if needed.

**List of problems**

1. Method CsvReader.Read(out string date, out string ticker, out string value) – should return string[]. It is not a level of business logic.
2. line.Split('\t'); - use separator as a method parameter
3. \_writer = new StreamWriter(fileName); - check file existing before access
4. \_writer = new StreamWriter(fileName); - use ‘using()’ for disposable objects
5. Use Disposable pattern for Open and Close functionality in your classes.
6. WriteReadCVS class rewrite as two classes. It is ‘Single responsibility principle’.
7. Use interfaces to inject some class to another.
8. Delete not required ‘using’ directives
9. Order ‘using’ directives. User libs directives should be written after standard libs directives
10. Check all unhappy cases and throw Exceptions with explanation messages.
11. Use ‘decimal’ type for money. ‘Double’ doesn’t have a needed precision for financial operations. In your code it is not bad, however other developers should cast your double to decimal for arithmetical operations to avoid problems.
12. When you parse string values you need to check for any exceptions and throw own exception with field name. In addition, an initial exception put to InnerException property.
13. Write unit tests for your exceptions.
14. Not duplicate code like you did with FakeQuoteRepository.
15. Write each class to own file
16. Do not use stubs like FakeQuoteRepository. Use Moq lib to create test objects from its interface. Set its functionality. Then inject to tested class.
17. Use test data as “strings” but not as file in general cases. It works faster.
18. Create test data for each case separately.
19. Creating architecture of classes you should think: what if next quote provider:
    1. give you .csv file with another structure, field order or separator
    2. give you .xls

So, you need to create an architecture with decoupled classes which could:

1. read/write data from storage
2. serialize/desirialize with Quote class
3. put/get to Irepository

Then you could assemble a new importer or exporter using different classes. Like LEGO ☺.