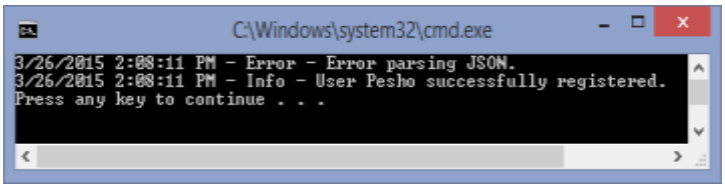
Homework: SOLID Principles in Software Design

This document defines the homework assignments from the "High-Quality Code" Course @ Software University. Please submit as homework a single **`zip`** / **`rar`** / **`7z`** archive holding the solutions (source code) of all below described problems.

Problem 1. Logger

Write a logging library for logging messages. The interface for the end-user should be as follows:

Sample Source Code	Sample Output

`ILayout simpleLayout = new SimpleLayout();`
`IAppender consoleAppender =`
`  new ConsoleAppender(simpleLayout);`
`ILogger logger = new Logger(consoleAppender);`
 	
`logger.Error("Error parsing JSON.");`
`logger.Info(string.Format("User {0} successfully registered.", "Pesho"));`
	

Library Architecture

The library should have the following components:

- **Layouts** - define the format in which messages should be appended (e.g. **SimpleLayout** displays logs in the format "**<date-time> - <report level> - <message>**")
- **Appenders -** responsible for appending the messages somewhere (e.g. Console, File, etc.)
- **Loggers** - hold methods for various kinds of logging (warnings, errors, info, etc.)

Whenever a logger is told to log something, it calls all of its appenders and tells them to append the message. In turn, the appenders append the message (e.g. to the console or a file) according to the layout they have.

Requirements

You library should correctly follow all **SOLID** principles:

- **Single Responsibility Principle** - no class or method should do more than one thing at once
- **Open-Closed Principle** - the library should be open for extension (i.e. its user should be able to create his own layouts/appenders/loggers)
- **Liskov Substitution Principle -** children classes should not break the behavior of their parent
- **Interface Segregation Principle** - the library should provide simple interfaces for the client to implement
- **Dependency Inversion** - no class/method should directly depend on concretions (only on abstractions)

Avoid code repetition. Name everything accordingly.

Implementations

The library should provide the following ready classes for the client:

- **SimpleLayout** - defines the format "**<date-time> - <report level> - <message>**"
- **ConsoleAppender** - appends a log to the console using the provided layout
- **FileAppender** - appends a log to a file using the provided layout
- **Logger**- a logger class which is used to log messages. Calls each of its appenders when something needs to be logged.

Sample Source Code

```
var simpleLayout = new SimpleLayout();

var consoleAppender = new ConsoleAppender(simpleLayout);
var fileAppender = new FileAppender(simpleLayout);
fileAppender.File = "log.txt";

var logger = new Logger(consoleAppender, fileAppender);
logger.Error("Error parsing JSON.");
logger.Info(string.Format("User {0} successfully registered.", "Pesho"));
logger.Warn("Warning - missing files.");
```

The above code should log the messages both on the **console** and in **log.txt** in the format **SimpleLayout** provides.

Extensibility

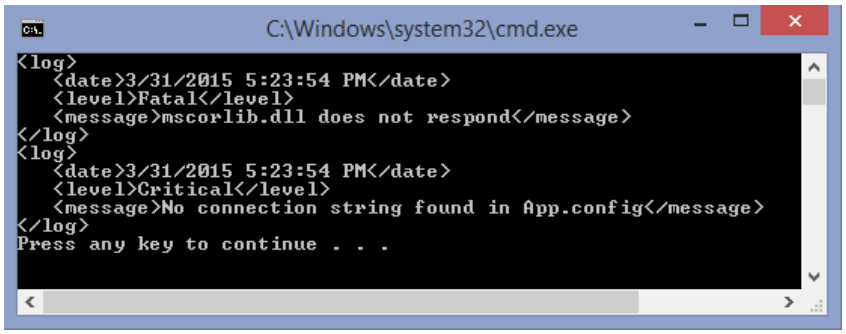
The end-user should be able to add his own **layouts/appenders/loggers** and use them. For example, he should be able to create his own **XmlLayout** and make the appenders use it, **without directly editing** the library source code.

Sample Source Code

```
var xmlLayout = new XmlLayout();
var consoleAppender = new ConsoleAppender(xmlLayout);
var logger = new Logger(consoleAppender);

logger.Fatal("mscorlib.dll does not respond");
logger.Critical("No connection string found in App.config");
```

Console Output



Report Threshold

Implement a **report level threshold** in all appenders. The appender should append only messages with report level **above or equal to** its report level threshold (by default all messages are appended). The report level is in the order Info > Warning > Error > Critical > Fatal.

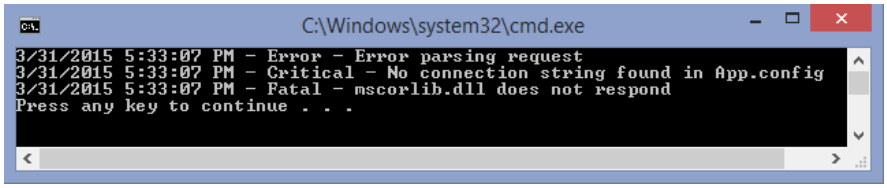
Sample Source Code

```
var simpleLayout = new SimpleLayout();
var consoleAppender = new ConsoleAppender(simpleLayout);
consoleAppender.ReportLevel = ReportLevel.Error;

var logger = new Logger(consoleAppender);

logger.Info("Everything seems fine");
logger.Warn("Warning: ping is too high - disconnect imminent");
logger.Error("Error parsing request");
logger.Critical("No connection string found in App.config");
logger.Fatal("mscorlib.dll does not respond");
```

Console Output



Only messages from error and above are appended.