# Exercises: Open / Closed and Liskov Principle

This document defines the exercises for ["Java OOP Advanced" course @ Software University](https://softuni.bg/java-basics-oop). Please submit your solutions (source code) of all below described problems in <https://judge.softuni.bg/Contests/265/Open-Closed-and-Liskov-Principle> .

## Logger

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

|  |
| --- |
| **Sample Source Code** |
| ***Layout*** simpleLayout = **new** SimpleLayout(); ***Appender*** consoleAppender = **new** ConsoleAppender(simpleLayout); ***Logger*** logger = **new** MessageLogger(consoleAppender);  logger.logError(**"3/26/2015 2:08:11 PM"**, **"Error parsing JSON."**); logger.logError(**"3/26/2015 2:08:11 PM"**, **"User Pesho successfully registered."**); |
| **Sample Output** |
|  |

**Logger** logs data and time (String) and a **message** (String).

### 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

Your 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 (You need to **implement** a **CustomFile** class) using the provided layout
* **LogFile** - a **custom file class** which logs **messages** in a **string builder** using a **method** write(). It should have a **getter** for its **size** which is the **sum of the ascii codes** of **all alphabet characters it contains** (e.g. a-z and A-Z).
* **Logger** - a logger class which is used to **log messages**. Calls each of its **appenders** when something needs to be **logged**.

|  |
| --- |
| **Sample Source Code** |
| ***Layout*** simpleLayout = **new** SimpleLayout(); ***Appender*** consoleAppender = **new** ConsoleAppender(simpleLayout);  ***File*** file = **new** LogFile(); ***Appender*** fileAppender = **new** FileAppender(simpleLayout); ((FileAppender) fileAppender).setFile(file);  ***Logger*** logger = **new** MessageLogger(consoleAppender, fileAppender);  logger.logError(**"3/26/2015 2:08:11 PM"**, **"Error parsing JSON."**); logger.logError(**"3/26/2015 2:08:11 PM"**, **"User Pesho successfully registered."**); |

The **above code** should log **the messages** both on the **console** and in **LogFile** 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** |
| Layout xmlLayout = **new** XmlLayout(); ***Appender*** consoleAppender = **new** ConsoleAppender(xmlLayout); ***Logger*** logger = **new** MessageLogger(consoleAppender);  logger.logFatal(**"3/31/2015 5:23:54 PM"**, **"mscorlib.dll does not respond"**); logger.logCritical(**"3/31/2015 5:23:54 PM"**, **"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** |
| Layout simpleLayout = **new** SimpleLayout(); ***Appender*** consoleAppender = **new** ConsoleAppender(simpleLayout); consoleAppender.setReportLevel(**ReportLevel**.***ERROR***);  ***Logger*** logger = **new** MessageLogger(consoleAppender);  logger.logInfo(**"3/31/2015 5:33:07 PM"**, **"Everything seems fine"**); logger.logWarning(**"3/31/2015 5:33:07 PM"**, **"Warning: ping is too high - disconnect imminent"**); logger.logError(**"3/31/2015 5:33:07 PM"**, **"Error parsing request"**); logger.logCritical(**"3/31/2015 5:33:07 PM"**, **"No connection string found in App.config"**); logger.logFatal(**"3/31/2015 5:33:07 PM"**, **"mscorlib.dll does not respond"**); |
| **Console Output** |
|  |

**Only messages** from **error** and **above** are appended.

### File

A file should write **all messages** **internally** and it should keep **information** about its **size**.

**Size** of a file is calculated by summing ASCII codes of all alphabet characters (a-Z). For example, a file appender with simple layout and a **single message** **"3/31/2015 5:33:07 PM - ERROR - Error parsing request" has size** 2606 (including all characters in **PM, ERROR, Error, parsing, request**). In case of **Xml layout**, the **file** would have **size** **6632**, because of the **extra characters** within the **tags**.

### Controller

Implement a **Controller** which reads **all appenders** that a **Logger** will have and **input messages** from the console. **Every message** should be **evaluated** by all the appenders and **logged** if they meet the report level. **Console appenders** should write **directly on the console**. **File appenders** write (save) the messages but **do not** print them.

### Input

On the **first line** you will get **N** - the number of appenders. On the next N lines, you will get **information** about the appenders in one of the **formats** below:

* **"<appender type> <layout type> <REPORT LEVEL>"**
* **"<appender type> <layout type>"**

If **no report level** **is provided**, the **appender** should be set **to record** all messages.

Next, until you get the "**END**" command you will receive **messages** containing **report level**, **time** and message separated by pipe "|":

* **"<REPORT LEVEL>|<time>|<message>"**

### Output

**Console appenders** should print directly at the **console** in the layout they are provided:

* **Simple layout example** - **"3/31/2015 5:33:07 PM - ERROR - Error parsing request"**
* **Xml** layout example (**date**, level and message **tags** are indented by **1 tabulation**) -

**<log>**

**<date>3/31/2015 5:33:07 PM</date>**

**<level>ERROR</level>**

**<message>Error parsing request</message>**

**</log>**

After the "**END**" command you should print **Logger info** which includes **statistics** about **every appender** (its **type**, **layout**, **report level,** **messages** appended and **file size** for file appenders):

**"Logger info**

**Appender type: <appender type>, Layout type: <layout type>, Report level: <REPORT LEVEL>, Messages appended: <count>, File size <size>"**

### Example

|  |
| --- |
| **Input** |
| 2  ConsoleAppender SimpleLayout CRITICAL  FileAppender XmlLayout  INFO|3/26/2015 2:08:11 PM|Everything seems fine  WARNING|3/26/2015 2:22:13 PM|Warning: ping is too high - disconnect imminent  ERROR|3/26/2015 2:32:44 PM|Error parsing request  CRITICAL|3/26/2015 2:38:01 PM|No connection string found in App.config  FATAL|3/26/2015 2:39:19 PM|mscorlib.dll does not respond  END |
| **Output** |
| 3/26/2015 2:38:01 PM - CRITICAL - No connection string found in App.config  3/26/2015 2:39:19 PM - FATAL - mscorlib.dll does not respond  Logger info  Appender type: **ConsoleAppender**, Layout type: **SimpleLayout**, Report level: **CRITICAL**, Messages appended: 2  Appender type: **FileAppender**, Layout type: **XmlLayout**, Report level: **INFO**, Messages appended: **5**, File size: **37526** |

## Blobs

This problem is originally from the [OOP-Exam-20-December-2015-Blobs](https://judge.softuni.bg/Contests/147/OOP-Exam-20-December-2015-Morning).

Blobs are slimy little creatures who have been at war for the last 300 years that have special abilities in the form of **behaviors** and **special attacks**.

You are given a partly finished library, which contains some **models** (Blob, Behavior and Attack). **Refactor** the given code and **complete** an application which supports **creating blobs** and **simulating fights** between them.

## Task 1 - Implement the Game Objects

A blob has a **name**, **health** and **damage**.

A blob also has a **behavior**. A **behavior** is triggered **when** a blob falls to **less or equal** to **half its initial health**. The following **behaviors** should be supported:

* **Aggressive Behavior** - doubles the blob's damage. **Each consecutive turn** the blob loses **5 damage**. The unit's damage cannot fall below its **initial value** (the damage before the **behavior** was toggled).
* **Inflated Behavior** - The blob gains **50 health**. **Each consecutive turns** the blob loses **10 health**.

A **behavior** can only be triggered **once**. It should be triggered **even if** the blob falls to **0 health**. If it is triggered a **second time**, an **error** should be raised.

A **blob** can **attack** another **blob**. The following attacks should be supported:

* **Putrid Fart** - the blob produces an attack with **damage** equal to its **own damage**
* **Blobplode** - the blob loses **half its current health** (e.g. from **55** health loses **27** health = **28** health left) and produces an **attack** with **damage** equal to **double its own damage**
  + The blob cannot fall below 1 health from attacking with Blobplode

A blob can perform an attack **multiple times** (only once per turn). A **blob** can have only a **single attack** (either **Putrid Fart**, **Blobplode** or any other attack) and a single behavior (either **Aggressive**, **Inflated** or any other behavior).

### Other Notes

* If a blob's attack **triggers a behavior**, the behavior should be applied **immediately** (i.e. a **behavior triggered by an attack** can affect the **attack** that triggered it)
* A **blob** should not fall below **0 health**
* **Dead blobs** cannot attack / be attacked

## Task 2 - Flexible Blobs

Design the blobs so they can work flexibly with **any behavior** and **any attack**.

## Task 3 - Improve the Models

Encapsulate **all internal behavior**. The implemented classes should **not reveal** any **internal logic**.

Avoid code repetition and promote code reusability by applying the good practices of OOP.

## Task 4 - Application Logic

From the standard input you will receive **commands**, each on a separate line. The application should support the following commands:

* **create <name> <health> <damage> <behavior> <attack>** - adds a new blob with the specified behavior and attack
* **attack <attacker> <target>** - forces a blob to perform an attack on another blob

The **attacking blob** produces an **attack** that deals damage to the **target blob's health**.

* **pass** - does nothing, skips the turn and progresses the game
* **status** - prints data about the current state of the game in the following format:

|  |
| --- |
| **Blob {name}: {health} HP, {damage} Damage**  **...** |

**Blobs** should be printed in **order of entry** in the game.

If a **blob** has been killed, the format should instead be:

|  |
| --- |
| **Blob {name} KILLED** |

* **drop** - ends the program

Each command should progress the game with **1 turn** after it is executed.

## Task 5 - Loose Coupling

The application should support the **creation** of blobs with **any behavior** and **attack**.

## Task 6 - Input / Output Independence

The application should be designed to work with **any input source** and **output destination**. In other words, it should **NOT** depend on the console.

## \* Bonus Task 7 - Blob Events

Implement a **fifth command**:

* **report-events** - if passed as **first command** in input the engine should **print detailed information** when **blobs attack each other**:
* When a **blob** toggles its behavior

|  |
| --- |
| **Blob {name} toggled {behavior-type}** |

* When a **blob** is killed (its **health** drops to 0 after all effects are taken into consideration)

|  |
| --- |
| **Blob {name} was killed** |

The **blobs** should **NOT** **directly** interact with the **engine** or any input/output classes.

This task is not part of the automated tests in the Judge system.

## Input

The input will be read from the standard input. On each line a command will be given (one of the described above).

## Output

The output should be printed on the console. Upon receiving the status command, print the current status of the game as described above.

## Constraints

* The **health** and **damage** will be valid 32-bit integer numbers
* The input will always **end** with the drop command
* The **report-events** command will always come **first** if present in the input

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| create Cenko 30 15 Inflated PutridFart  create Boko 50 20 Aggressive Blobplode  attack Boko Cenko  status  status  status  status  status  status  drop | Blob Cenko: 50 HP, 15 Damage  Blob Boko: 25 HP, 40 Damage  Blob Cenko: 40 HP, 15 Damage  Blob Boko: 25 HP, 35 Damage  Blob Cenko: 30 HP, 15 Damage  Blob Boko: 25 HP, 30 Damage  Blob Cenko: 20 HP, 15 Damage  Blob Boko: 25 HP, 25 Damage  Blob Cenko: 10 HP, 15 Damage  Blob Boko: 25 HP, 20 Damage  Blob Cenko KILLED  Blob Boko: 25 HP, 20 Damage |

|  |  |
| --- | --- |
| **Input** | **Output** |
| create Fiki 90 5 Inflated Blobplode  create Jorjo 30 25 Inflated Blobplode  attack Fiki Jorjo  status  attack Fiki Jorjo  status  drop | Blob Fiki: 95 HP, 5 Damage  Blob Jorjo: 20 HP, 25 Damage  Blob Fiki: 33 HP, 5 Damage  Blob Jorjo: 60 HP, 25 Damage |

|  |  |
| --- | --- |
| **Input** | **Output** |
| create Sir 70 20 Aggressive Blobplode  create Stenly 33 15 Aggressive Blobplode  create Royce 50 20 Inflated Blobplode  status  attack Stenly Royce  status  status  drop | Blob Sir: 70 HP, 20 Damage  Blob Stenly: 33 HP, 15 Damage  Blob Royce: 50 HP, 20 Damage  Blob Sir: 70 HP, 20 Damage  Blob Stenly: 17 HP, 15 Damage  Blob Royce: 70 HP, 20 Damage  Blob Sir: 70 HP, 20 Damage  Blob Stenly: 17 HP, 15 Damage  Blob Royce: 60 HP, 20 Damage |

|  |  |
| --- | --- |
| **Input** | **Output** |
| report-events  create Petya 20 10 Aggressive PutridFart  create Emi 30 15 Inflated PutridFart  attack Petya Emi  attack Petya Emi  attack Emi Petya  attack Emi Petya  pass  status  drop | Blob Emi toggled InflatedBehavior  Blob Petya toggled AggressiveBehavior  Blob Petya was killed  Blob Petya KILLED  Blob Emi: 30 HP, 15 Damage |