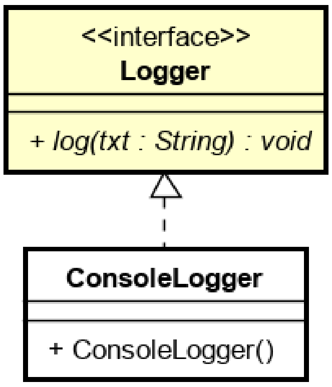
Exercise Logger – Proxy design pattern

You have a simple Logger class:

```
public class ConsoleLogger implements Logger
{
  public ConsoleLogger() {}

  public void log(String txt)
  {
    System.out.println(txt);
  }
}
```



with interface `Logger` defined this way:

```
public interface Logger
{
  void log(String txt);
}
```

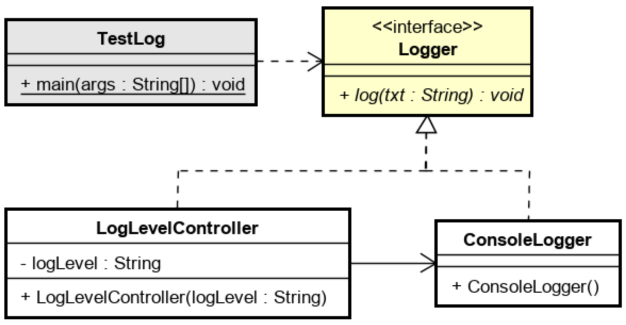
Sometimes companies might want to set log-levels, to control how much is output to the log. We can create such a filtering using the proxy pattern. The idea is that any relevant change to log-filtering happens in the proxy. Say you have two log-levels:

- verbose
- sparse

When the level is set to 'verbose', everything will be put in the log. If you can come up with more filters, you are welcome to expand the exercise. When the level is set to 'sparse' only messages containing the word 'error' should be put in the log. You may find use of the String methods called "toLowerCase" and "contains".

Use the Proxy design pattern to change your Log solution, so that the 'proxy log' class handles filtering. This means, you should have a setup similar to the below:

Have the `LogLevelController` (the proxy class) implement interface `Logger`. Make the proxy handle the filtering.



TestLog
+ main(args : String[]) : void
<<interface>>
Logger
+ log(txt : String) : void
LogLevelController
- logLevel : String
+ LogLevelController(logLevel : String)
ConsoleLogger
+ ConsoleLogger()

Test the solution having a client class (class `TestLog` with a `main` method) with a reference to the interface and an object created as a `LogLevelController`. (*Note that you are not allowed to have a variable of type `LogLevelController` in order to follow the Proxy design pattern*). Add a few logs to see if it is working as expected. Try also to test 'sparse' logging and call log with texts including the string "error" and some without.