## **Key Word In Context – Singleton Wrappers**

Change Input to read input in from the console instead of from a file. Change Output to output the result to the console instead of to a file. Don't worry about exception handling or incorrect/invalid user input.

#### Design requirement:

- 1. Keep the Java layer abstracted from your code.
- 2. ScannerWrapper should be a singleton that sets up a Scanner and has a nextLine() method to leverage Scanner's nextLine() method.
- 3. SystemWrapper should be a singleton that has a println() method to leverage System.out.println()
- 4. Dependency inversion dictates that the singletons should be passed in to the classes that need them. Input needs both. Output needs only SystemWrapper. For this assignment, pass them in through the method, not setters or constructor (we will change this in the next assignment). Since MasterControl needs both Input and Output, MasterControl also needs both singletons. This means that something needs to pass them in to MasterControl (the main method should).

#### **Guidelines:**

```
MasterControl:
       Method signature change:
       public void start(ScannerWrapper scannerWrapper, SystemWrapper systemWrapper)
       Main method, given to you for free:
       public static void main(String[] args) throws IOException {
            MasterControl masterControl = new MasterControl();
            masterControl.start(ScannerWrapper.getInstance(), SystemWrapper.getInstance());
        }
Input:
       Method signature change:
       public List<String> read(ScannerWrapper scannerWrapper, SystemWrapper
                                                                    systemWrapper)
CircularShifter:
      Unchanged
Alphabetizer:
       Unchanged
Output:
       Method signature change:
       public void write(List<String> lines, SystemWrapper systemWrapper)
```

The program should prompt and work exactly as in the screenshot below:

Please enter lines to add, then enter -1 to finish:
Sense and Sensibility
Architecture Software
Crouching Tiger Hidden Dragon
-1
and Sensibility Sense
Architecture Software
Crouching Tiger Hidden Dragon
Dragon Crouching Tiger Hidden
Hidden Dragon Crouching Tiger
Sense and Sensibility
Sensibility Sense and
Software Architecture
Tiger Hidden Dragon Crouching

### For the included tests:

Make sure the test source folder is already created (like in A1). Then:
Right click on your Project in Package Explorer -> Configure -> Convert to Maven Project
Group ID: firstname.lastname all lowercase
Artifact ID: assignment name all lowercase

Double click on "pom.xml" in your project Click on "pom.xml" on the lower-right of the screen

CTRL + SHIFT + F will format the file and also give you feedback that there are no semantic errors in the POM file. CTRL + S will save the file an automagically import Mockito into your project.

```
= □ 🖺 A2 - Singleton/pom.xml 🛭
1@roject xmlns="http://maven.apache.org/POM/4.0.0"
 ≅ А1
                                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
  ➤ JRE System Library [JavaSE-11]
                                              <groupId>boris.valerstein</groupId>

✓ 

Ø src

                                               <artifactId>assignment2</artifactId>
    # (default package)
                                              <version>0.0.1-SNAPSHOT</version>
       > Alphabetizer.java
                                                  <sourceDirectory>src</sourceDirectory>
       > DircularShifter.java
                                       10
                                                  <testSourceDirectory>test</testSourceDirectory>
      > 🛭 Input.java
                                                  <plugins>
                                       11⊜
      > 🛭 MasterControl.java
                                       12⊜
                                                       <plugin>
       > 🛭 Output.java
                                       13
                                                            <artifactId>maven-compiler-plugin</artifactId>
                                       14
                                                            <version>3.8.0
       > 

ScannerWrapper.java
                                                           <configuration>
       > 

SystemWrapper.java
                                       16
                                                               <release>11</release>
  17
                                                           </configuration>

→ # (default package)
                                      18
19
                                                       </plugin>
                                                   </plugins>
       AlphabetizerTest.java
                                      20
21<sup>®</sup>
                                              </build>
       > 🛽 CircularShifterTest.java
                                              <dependencies> 3
       > 🗓 InputTest.java
                                                  <dependency>
       > A MasterControlTest.java
                                       23
                                                       <groupId>org.mockito</groupId>
                                                       <artifactId>mockito-core</artifactId>
       > 🚇 OutputTest.java
                                       25
                                                       <version>3.3.3
  > 🛎 JUnit 5
                                                       <scope>test</scope>
  > Maven Dependencies
                                       27
                                                  </dependency>
  > 🗁 bin
                                              </dependencies>
    target
                                      29 </project>
    kwic_output.txt

    pom.xml
    1

                                  > Overview Dependencies Dependency Hierarchy Effective POM pom.xml
```

Remember that tests are an indicator that your code works as expected, but always run the code from the main method as well to make sure the program runs correctly for real.

If the Wrappers are incorrect, the JUnit tests may pass, but you won't get full credit for the assignment.

# **Submission**

Same as A1.

Rubric:

Functionality – 50 points

This includes following directions.

Wrappers + Singletons – 40

Correctly implemented

Clean Code – 5 points UML – 5 points