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:

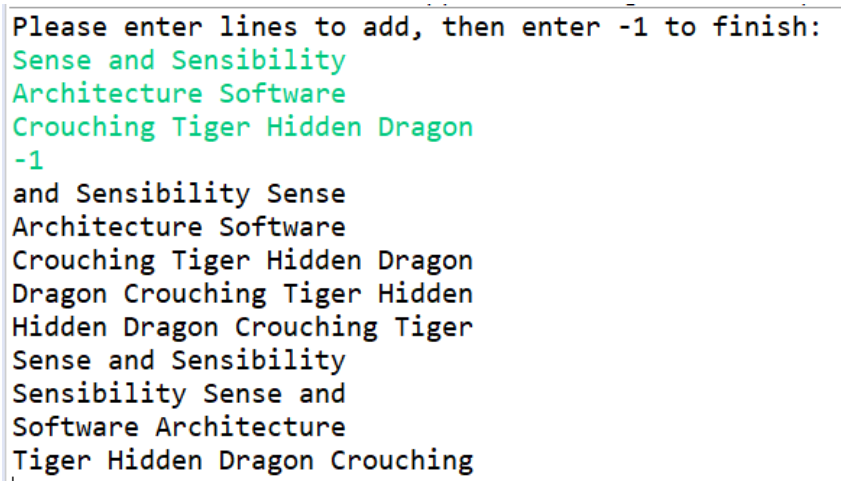
Unchanged

Output:

Method signature change:

```
public void write(List<String> lines, SystemWrapper systemWrapper)
```

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



For the included tests:

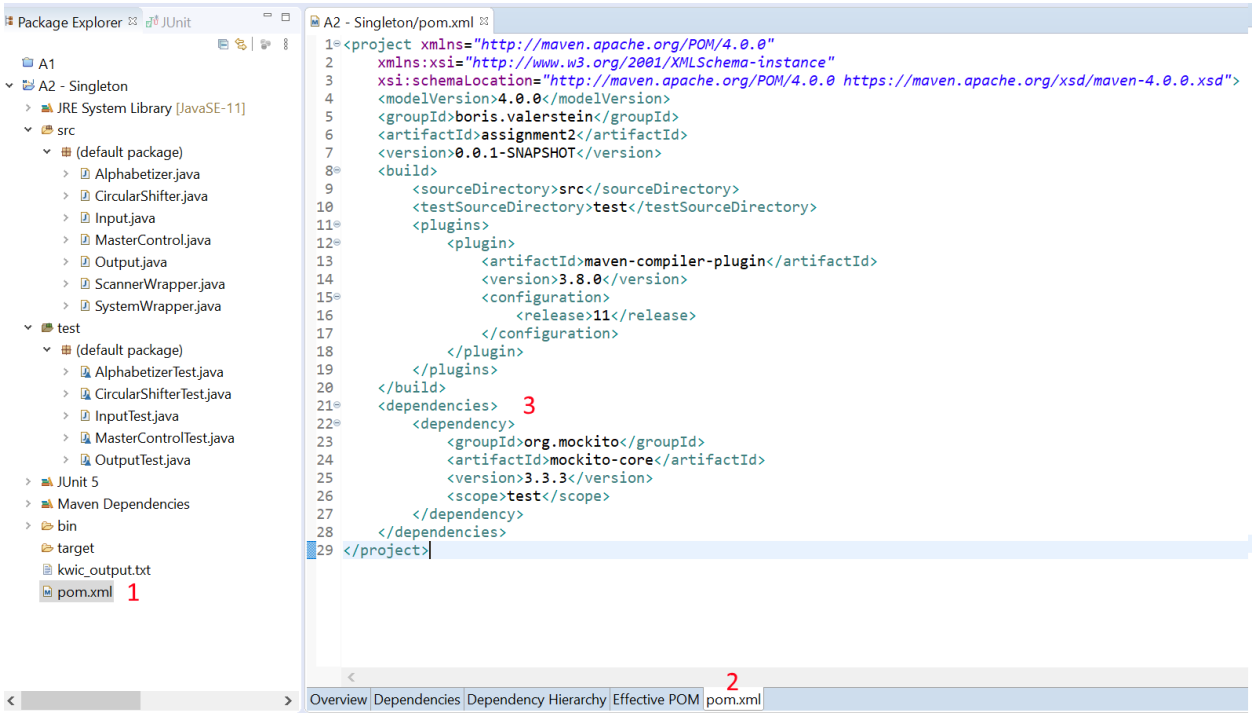
Make sure the test source folder is already created (like in A1). Then:
Right click on your Project in Package Explorer -> Configure -> Convert to Maven Project
Group ID: firstname.lastname all lowercase
Artifact ID: assignment name all lowercase

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

Before `</project>` paste in:

```
<dependencies>
	<dependency>
		<groupId>org.mockito</groupId>
		<artifactId>mockito-core</artifactId>
		<version>3.3.3</version>
		<scope>test</scope>
	</dependency>
</dependencies>
```

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.



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