SoftGel Pills 1:

Introduction

Table of Contents

From the Execs

For your first assignment, we would like you to create two types of pills: our pain reliever pill, AcheAway, and our insomnia-fighting pill, Dreamly. Each pill should have a name, strength in mg, size in mm, color, casing material, aqueous solution, and active ingredient. We manufacture the pills by first creating a casing, then injecting an aqueous solution and active ingredient into the casing. Here are the specs for AcheAway and Dreamly.

	Casing	**Solution**	**Active**
AcheAway	Gelatin	Saline	Acetaminophen
Dreamly	Plasticizer	Oil	Zolpidem

From your Manager

Based on the requests from the executives, it sounds like we should write three classes: GelCap, AcheAway, and Dreamly. We can abstract all generic pill data into the GelCap class and have the others inherit from it. In the constructor, set the basic fields and then run the manufacturing process, which we should separate into its own method. We should also make a demo class and unit tests for every file. Put the demo class in a separate package, but not with the unit tests.

Project Setup

Repository

To set up the project you should clone your shared repository on the student machine. Each group member should have their own copy. Create a directory inside of src/main/ named pills. This will be the name of our root package. Create a directory named pills in src/tests/ as well so that our unit tests can be in the same package. Create a directory named client in src/main/ which is where the demo class will go. Finally, edit the readme file. Add a general description of the project, your team name, and your team members. If you want to learn more about styling a markdown (md) file, check out this reference on markdown syntax here.

Group

Create a document in the docs directory named Sprint1-Planning.txt. You could use a different type of document if you want to put in a little more work to make the document look nice, such as a pdf, but this is purely optional.

Your group will be organized into two subteams. These subteams are not permanent for the whole semester, just this iteration of the project. The subteams should either have 2 people in each or 2 in one and 3 in the other, depending on your group size. In the

Sprint1-Planning file, add some text that says which members are in which subteam.
In Sprint1-Planning, add a section about what agile practices your team is going to use. Add SCRUM Methodology and Pair Programming to this section. Also add any other Agile practices or artifacts that your group would like to use, such as a KanBan Board or Test-First Programming.

Finally, you will be performing a retrospective at the end of the project. As a group, select a retrospective template from here. You will not fill it out until the end of the project, but it can be helpful to know what sorts of questions you will be answering at the end. At the end of the project, if you decide a different template would be better, you will be free to switch. You will also not need to use the actual template document if you don't want to, as long as you answer the same questions. For now, just make a decision together so that you have the questions in your mind. You do not need to add anything to your repository yet.

Stories

The work for this project has already been split up into stories for you and each story has been assigned to a subteam. The stories do not necessarily need to be done in order. Note that a different team writes the tests for a class than the team that writes the class. Also, both subteams write tests and source code.

- Story 1 (Subteam A): Create tests for the GelCap class
- Story 2 (Subteam B): Create the GelCap class
- Story 3 (Subteam B): Create tests for the AcheAway Class
- Story 4 (Subteam A): Create the AcheAway subclass
- Story 5 (Subteam A): Create tests for the Dreamly Class
- Story 6 (Subteam B): Create the Dreamly Class
- Story 7 (Subteam A and/or B): Create a Demo class.

Each story should be on its own branch and, when they are completed, the subteam that wrote the code should create a pull request on GitHub for that story branch. The other subteam should review the code and, if it is all good, confirm the pull request and merge the branch. If there are any issues with the code, use the comment feature on the Pull Request to discuss it.

Detailed Class Descriptions

For all classes, see the UML diagram for basic structure and information

GelCap

- An abstract pill class from which all pills produced by SoftGel Pills for Health will inherit.
- The constructor should set the name, strength, size, and color fields to values passed in. Then it should call manufacture.
- All of the fields should have getters. None should have setters.
- The manufacture method should print "Manufacturing ... \n" and call addCasing, addSolution, and addActive, in that order, then print " ... completed manufacturing\n".
- The addCasing, addSolution, and addActive methods should be abstract.
- toString should return a string stating the pill’s strength and name in the form "<strength>mg <name> Pill" with the number in strength having 2 decimal places.
- description should return a string stating all the properties of the pill in the following form. Every line but the first begins with a tab character and every line but the last ends with a newline character. Every double should have 2 decimal places.

```
<name> Pill
    Strength: <strength>
    Size: <size>
    Color: <color>
    Casing: <casing>
    Solution: <solution>
    Active: <active>
```

AcheAway

- Subclass of GelCap
- The constructor should accept a strength, color, and size and call the super constructor with the name "AcheAway" and the values passed in.
- The addCasing, addSolution, and addActive methods should be overridden so that they set their corresponding field to the correct value (see spec chart from the execs). They should also print "adding x y" where x is the value being put into the field and y is the name of the field. For example, addSolution would print "adding saline solution\n".

Dreamly

- Subclass of GelCap
- The constructor should accept a strength, color, and size and call the super constructor with the name "Dreamly" and the values passed in.

- The addCasing, addSolution, and addActive methods should be overridden in the same way described in AcheAway, but should use the values for Dreamly from the specs table.

Demo

- This file should be in a different directory named client.
- It will need to import the pills classes since it is not in the same package
- It should contain a main method and nothing else.
- The main method should create one of each type of pill, print them, and print the output of their description method.
- You should be able to run it using the `make demo` command.
- At the end of the instruction there is an example of what the output should look like. You can use this as a manual means of testing.

Detailed Test Descriptions

Each of the three classes should have a corresponding test file in src/tests/pills.

GelCapTest

GelCapMock

- The first thing that we need to do is create a mock of GelCap. Since GelCap is abstract, we cannot instantiate a GelCap object. Create a private inner class named GelCapMock to use in the tests.
- GelCapMock needs to extend GelCap
- It should have a constructor that looks the same as the GelCap constructor and simply passes all the arguments through to the super constructor.
- It should override the three abstract methods. Normally, we would just make them empty, but let's add a little bit of code to give us greater testing capabilities.
- Have addCasing set casing to "X" and print "X\n"
- Have addSolution set solution to "Y" and print "Y\n"
- Have addActive set active to "Z" and print "Z\n"
- The addition of these 6 statements will let us test the getters for these three fields and test that the manufacture method is calling these methods. Normally we would use a spy to check that these methods were

called, but since we are not using Mockito, we can't do that.

Test Utilities

- You will need test values for name, strength, size, color, casing, solution, and active. Create these as constants at the top of the file. You can use any values for them except casing, solution, and active should be the same as the values used in the private class ("X", "Y", and "Z").
- We will need to redirect System.out to test print statements, so add the fields necessary for this along with the appropriate statements in your beforeEach and afterEach methods.
- Create the getOutput private helper method for use when redirecting System.out.
- Create a GelCap field for your testing object, and initialize it to a GelCapMock object in your beforeEach. Use the constants you created as the arguments to the constructor.

Tests

- There should be 10 Tests
- testName, testColor, testSize, testStrength
  - Test that the values passed to the constructor are the same as the values returned by the getter methods.

        - This is simultaneously testing that the constructor works and that the getters work.
    - testCasing, testSolution, testActive
        - These tests check that the constructor called manufacture, that manufacture called the corresponding add method, and that the getter method returns the value stored in the appropriate field.
    - testToString
        - For this test, you should add another constant named TOSTRING_FSTRING and set it equal to the format string that GelCap should be using. Then, in the test, use String.format and the appropriate other constants, to check that toString outputs a properly formatted string.
    - testDescription
        - This test works the same as the test for toString, except that you need a different constant (call it DESCRIPTION_FSTRING) for your format String.
        - It is also more complicated and involves more variables, but it works the same way.
    - testManufactureProcess
        - This test will use getOutput to test that the manufacture process has the correct print statements. It will also test that the correct methods get called in the correct order since we added print statements to them.

        - You should use a MANUFACTURE_FSTRING constant and use it the same way you used the DESCRIPTION_FSTRING and TOSTRING_FSTRING

AcheAwayTest

- The general setup of this test file is the same as that of GelCapTest, including the constants, lifecycle hooks, and helper methods. However, we do not need a mock class.
- You can use the same tests for testStrength, testSize, and testColor as were used in GelCapTest. See the descriptions in the GelCap section for more details. However, it should be noted that we are really only trying to use these to test the constructor this time, since we do not need to test the getters since AcheAway does not override these methods.
- testName can also be the same, but we should change the name of the constant from TEST_NAME to CORRECT_NAME since we are no longer supplying our own name.
- You can use the same tests for testCasing, testSolution, and testActive as were used in GelCapTest. However, we again want to change the constants to CORRECT_FIELD from TEST_FIELD, since the AcheAwayTest have specific values they should be storing in these fields and we are not supplying our own through a mock.
- There do not need to be tests for description or toString since these methods are entirely untouched.

- testManufactureProcess can be the same as the test in GelCapTest, but the MANUFACTURE_FSTRING constant will need to be changed.

DreamlyTest

- This test file will essentially be the same as AcheAwayTes but with different values for some constants and using a Dreamly object instead of an AcheAway object.

Wrapping Up

- Fill out the retrospective template that you chose at the beginning of the project. If you think a different template would be better, you can change it first. Put the retrospective in the docs directory with the name Sprint1-Retro. You can use a word document, pdf, or simply create a text file and write each of the questions above your answers.
- Make sure that everything has been pushed to GitHub and that it has been merged into main.
- Your code should look clean and consistent, have adequate documentation (ie comments / JavaDocs), and pass checkstyle.
- Make an "Introduction" branch when you are finished that branches off of main to save this iteration of your project. This is the branch that will be graded.
- Grade Breakdown

    60 pts: Code correctness (including tests)

    20 pts: Git Usage

    10 pts: Agile practices

    10 pts: Appearance / Checkstyle

Example Demo Output

```
bash-4.2$ make demo
java -cp .:bin client/Demo
Manufacturing...
adding gelatin casing
adding saline solution
adding acetaminophen active
...completed manufacturing
2.00mg AcheAway pill
AcheAway Pill
        Strength: 2.00
        Size: 5.00
        Color: yellow
        Casing: Gelatin
        Solution: Saline
        Active: Acetaminophen
Manufacturing...
adding plasticizer casing
adding oil solution
adding zolpidem active
...completed manufacturing
1.00mg Dreamly pill
Dreamly Pill
        Strength: 1.00
        Size: 7.00
        Color: white
        Casing: Plasticizer
        Solution: Oil
        Active: Zolpidem
```