SoftGel Pills 3: Singletons and Proxies and Visitors, Oh My!

Table of Contents

From the Execs

Your internship is almost over! You just have about one month left. You have done a great job and we are very close to having a great system. There are just a few more things for you to implement.

First, we are having an issue with the pill component generators. They are VERY resource intensive. Our system slows down dramatically when they run. We bought another robust server and would like it if you could move those functions to the new server.

Second, we have had some issues with multiple stores making multiple factories. We only have one of each factory so every store should use the same one. This has been causing some major problems, so I hope you can figure it out.

Finally, we want some extra checks and functions in our store. We need to make sure that child and adult pills are not mixed in an order and we need to check that an order doesn't have too many failed pills in it. Also, we would like the users to be able to check the total strength of each type of pill in their order before logging out.

From your Manager

Wow, they are asking a lot of you for your last month. Let's take each request one by one. I also have a few suggestions on improving the system so that it will be more robust after you leave.

They first asked you to move the generation methods to a different server. This is called distributed computing and is a great thing to do. We can use the remote proxy pattern for this. Java has an RMI (remote method invocation) system that we can use. It shouldn't be too difficult for you.

Second, they want you to make sure there is only one of each factory. That is exactly what the singleton pattern is for. You will need to make it threadsafe. I would recommend using method level synchronization, but you could use double-check locking if you want. Unfortunately, the singleton pattern is hard to abstract in Java, so we need to put it in both concrete factories rather than the abstract factory.

Third, they are asking for a bunch of different functions in the store that involve examining pills in the current order. The visitor pattern would work well here and would make it easy for new functions like these to be added in the future. You will need three different visitors, one for each function.

Finally, I think we should stop using null to represent failed pills. We don't want to get NullPointerExceptions or have to constantly check if a pill is null. Use the NullObject pattern to fix this problem.

Project Setup

Repository

You will be using the same repository that you used in Part 2. Make sure you have made the ProductionAndVariations branch for the previous project iteration before you start.

Group

You will have the most freedom you have had yet in terms of organizing your group. Here is a list of some things that you need to decide.

- How do you want to do subteams? You can split them up however you want, but make sure it helps encourage agile practices.
- How many sprints do you want?
- How long do you want your sprints to be?
- What stories do you want to make? You are making the entire product backlog this time. Use the UML and class descriptions to help.
- How do you want to divide up the stories?
- How do you want to do sprint reviews? They need to have some sort of output that you can show me.
- How do you want to do sprint retrospectives? They need to have some sort of output that you can show me.
- Document all your decisions in the docs folder.
- Problem solve together. You will have less direction.

Class Descriptions

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

GelCap

- Update with the abstract accept method

AcheAway, Dreamly

- No changes

ChildAcheAway, ChildDreamly, AdultAcheAway, AdultDreamly

- Update with implementations of the accept method

NullDreamly, NullAcheAway

- Create based on UML.
- Use falsy values in the super constructor
- Implement accept like normal

CasingGenerator, SolutionGenerator, ActiveGenerator

- Update to match the UML

The 6 Generator Classes

- Update to match the UML
- The constructor should pass the port to the superconstructor
- The constructor and method should throw

GelCapRecipes

- Change this to access the generator classes remotely.
- NOTE: You are not actually using your generator classes. I am hosting the same classes that you just made on another machine since you don't have a good way to put your versions on another machine. Do NOT use new anymore to create the generators.
- Use Naming.lookup to get the generators
    - The ip is 152.10.10.40
    - GelatinCasing - port 1998
    - PlasticizerCasing - port 1997
    - OilSolution - port 1996
    - SalineSolution - port 1995
    - AcetaminophenActive - port 1994
    - ZolpidemActive - port 1993

GelCapFactory

- Update the produce methods to deal with the potential remote errors that could occur when using the remote generators

AdultGelCapFactory, ChildGelCapFactory

- Update to follow the UML diagram
- getInstance
    - Follow the singleton pattern
    - Make it thread safe using either method level synchronization or double check locking

Inspector

- Create based on the UML
- All of the inspect methods should have an empty implementations
- The other two methods are abstract

ConsistencyInspector, FailureInspector, StrengthInspector

- Create based on the UML
- You should be able to figure out what the methods should do based on their names and your understanding of the visitor pattern
- Report just needs to return a string explaining the current state

SoftGelPillStore

- Update the store to match the UML diagram
- getDreamlyStrength and getAcheAwayStrength should use a StrengthInspector to calculate the strengths of the current order and return the appropriate value
- printCurrentOrder should do exactly what it says.
- tooBigFailRate should check the failRate value passed to it and see if it is too big. Remember 10 should be the average. You can decide how big is too big.
- checkFailRate should use a FailureInspector to calculate the failure rate of the current order and return the calculated failRate.
- consistentOrder should use a ConsistencyInspector to check if the current order is consistent and return true if it is, false if it isn't.
- checkOut will need to make some additional checks before returning the pills
    - Check that order is consistent. If it is not, print a message and return null.
    - Check that the fail rate is not too big. If it is, print a message and return null.
    - Optionally, you may add functionality to help the user resolve these problems.

Demo

You can no longer use my demo! You will need to create your own. You can use the previous demo as a template, but you will need to have the demo include the new functionality that was added (such as letting the user call the other public store methods.

The system is your oyster here. There are tons of ways you could write a demo to play with this system. Writing a great demo will be a major factor in one award (see awards section). Having a very bad demo could lose you points. The demo should not be less comprehensive than the previous one and it should include at least a couple additional functions.

Test Descriptions

Make decisions as a group about tests. Everything else should be tested and previous tests may need to be updated. Here are some tips for getting a good grade on testing.

- You do not need to test anything that is private.
- You do not need to test the proxy functionality since I have put that on another machine. You can still test the generators by manually creating them in the tests rather than getting them from the server.
- You do not need to test your demo
- You should test that you can't make multiple factories
- SoftGelPillStore should have a lot of robust tests. Try to test as many scenarios as you can think of.

Awards

There are three awards that will be given at the end of this project. The winners will have their group names and individual names (if given permission) on my website where their achievements will be immortalized. They will also get 15pts refunded from any points taken off. Here are the awards.

1. Best Program
    a. An excellent demo is the main component for this award.
    b. Any additional functionality that is added will also be considered (such as having a way for the user to rectify having an order will null pills in it).
2. Best Tests
    a. The most comprehensive and useful unit tests are the main component for this award.
    b. If you implement integration or end-to-end testing, that would be a huge boost to potentially winning.
3. Best Code Style
    a. This award is all about style. To win this you will want to have very nicely formatted code, comprehensive documentation, and efficient, well-structured code.

Wrapping Up

Take the Project Evaluation quiz on AsuLearn. This must be done before the first day of exams.

Reminders

- Do your retrospectives and reviews.
- Make sure you have documented how you are working in an agile manner.
- Make sure everything is on main.
- Clone your repo in a new area, run make compile, make test, make demo, and make check to make sure it works (this is exactly what I will do when grading).

Participation

Your individual participation will also be considered. If you did not participate in your team's project or did very little, your grade will be docked points. It is theoretically possible for your teammates to get full credit and you get no credit if they did all the work. Participation will be evaluated based on 3 things: teammate evaluations, GitHub activity, attendance on workdays. Additionally, teammates who went above and beyond may have their grades boosted.

Grade Breakdown

Total:	400 points
Source Code	100 points
Testing	100 points
Git Usage	80 points
Agile Practices	80 points
Appearance / Checkstyle	40 points