Assignment 6a/b - Design Patterns

Version: April 16, 2020

General

This assignment is a big assignment and is meant to be for 2 weeks overall. The assignment describes everything you need to do it basically introduces Design Patterns while also incorporating everything you have learned so far. THIS ASSIGNMENT WILL TAKE TIME!

You will work on GitHub and I do want to see that you work and commit in a good way and not just work on the very last day. I also want to see a clean, well organized Git repository (eg. good gitignore and all that).

You need to create a PDF document (specifics below) for your Deliverable which you will submit on Canvas.

Make sure you stick to the requirements in this assignment and to include all necessary information in your PDF. We will not reward points later on if you forgot to submit something (this includes the PDF). Make sure you clone your repo at the end and test if it runs. to make sure you have put all necessary files in the repo. This is a good practice anyway, so you are sure what you submit actually works.

This assignment is split up into 2 deliverables to force you to work continuously and not just in the last couple of days. See a little later what you need to submit for each deliverable.

The specifics

1. Work on Git and GitHub for this assignment (on a private repo, add ser316asu as collaborator), commit often we want to see the different stages you were in (it is ok if the first versions are partial, look bad etc). Include an appropriate gitignore file into your repository. Make sure you use good commit messages and group your commits well. (7 points)
2. Each Design Pattern below is worth 15 points (45 points) :
   - You need to create 3 design patterns from the Gang of Four for the requirements below.
   - You have two choices:

a) The minimal requirement for full points: Have the Design Pattern implemented independently from each other (the patterns should not "work together"). Basically, each one of them can be run separately. In this case you would implement each pattern in a separate package and in a Main class first show the functionality of one pattern then the other etc. This is the easier version, since the DP are not intertwined which many struggle with.

b) Getting full points plus some extra credit points: Implement your code with the Design Pattern being intertwined (you can of course still have packages if you like), working together and forming one good project, while the DP still need to be implemented correctly and make sense.

- Make sure you comment your code well, so we see what you are doing or tried to do.
- You do NOT need to meet all the functional requirements specified in Task 1, but each Design Pattern should at least implement 3 requirements (not all might fit into your Design Pattern but might be side functionality). In your PDF describe which requirements you fulfilled in each DP and also make sure you comment on it in your code. If you chose version 2 above, then 9 requirements need to be fulfilled overall in your code.
- The code needs to be fully functional and needs to "do something" (there needs to be a basic simulation that shows the functionality of the 4 requirements you chose. Just creating the skeleton of the Design Pattern with no data is NOT sufficient. I want to see a running functional implementation that does fulfill at least 4 requirements and runs.
- Make sure your Main does have good print outs so we and you can see what is going on.
- We will grade you based on the code that you've written. (eg. Is the pattern implemented correctly? Does your code fulfill at least 3/9 requirement? Is the code well written? Does the Main run this pattern? Is the project structure good and correct? Does the chosen Design Pattern make sense? Is your repository clean?)
- Minimal Requirement for each package: There is some kind of simulation included in Main that includes farmers, animals and/or crops (so you need to instantiate some this can be hard coded or random), some affinities and has day and night cycles. The simulation ends when the farm goes bankrupt or reaches a certain upgrade level or currency threshold.

3. Create a Gradle file - requirements (**All the below Gradle requirements are mandatory, you will not get points for this whole task if this does not work**). We will not import anything into an IDE.
   - Your code HAS TO execute via "gradle run" via command line (mandatory)
   - Include Checkstyle and Spotbugs (mandatory)
   - Include JUnit 4 (mandatory)

    - Creates an executable jar file (not mandatory)

4. Checkstyle and Spotbugs: make sure you adhere to coding standards you set (in your Checkstyle XML) and that Checkstyle and SpotBugs do not show anything. If you cannot get rid of everything you should mention this in your PDF document and explain why you were not able to fix it. In your PDF include a screenshot of your Spotbugs and Checkstyle report. You will not get points for this if there are no screenshots that show us your reports. (10 points)

5. TravisCI: Set up TravisCI so it builds your project, runs your Unit Tests and also Spotbugs and Checkstyle. (5 points)

6. Include JUnit tests (again, it must be JUnit 4) and test your code. You should reach at least 80% code coverage (excluding your Main, getters and setters). Include a screenshot of your JUnit and jacoco report, showing your tests pass and your code coverage. You will not get points for this if there are no screenshots that show us your reports.(15 points)

7. Include a short screencast (does not have to have audio) showing you doing a git clone from your repo, gradle build, showing your reports (JUnit, Spotbugs, Checkstyle, TravisCI), and gradle run (max. 2min). (8 points)

8. README.md on GitHub (well written and easy to find everything)
    - Include a link to your screencast
    - Explain each of your Design Patterns briefly and mention which of the requirements you fulfilled with this Pattern. (5 points)

9. You can earn up to 15 points extra credit, see at the end of this assignment for specifics.

IMPORTANT: If your Code has compile errors you will receive 0 points. Make sure your Main runs without errors. Example you are only able to make 2 of the 3 Design Pattern run in your Main and the 3rd is implemented but does not work. Then you should not include number 3 in your Main. You will get credit for the the two DP and will get partial credit for number 3 (if it has no compile errors).

As stated before you do not have to implement all the requirements and you can choose any 3 design patterns from the Gang of Four for task 1 that you see fit. The HINTS might help you (they are not requirements). You are relatively free of how to do things. Please mark where you see your Design Patterns in your code. This is coding extensive again, start early it will take a while to figure things out.

Requirements

Below are some of the functional requirements for the application.

Super Landwirtschaft Universum

- A new world must start with at least 1 farm.

- Farms can be of different types, such as an animal farm, a crop farm, a hybrid farm and so on. You can choose to make something up too.
- The simulation should run on cycles. A cycle is considered to be of 2 parts - 1 day time and 1 night time.
- Passive currency is earned with each new day (not night). This passive currency income is generated from selling crops or animal products, or both depending on your farm.
- More currency can be made from farmer, animal or crop affinities (examples below).
- Farms are automatically upgraded once the farm has acquired enough currency. This could mean that the farm is expanded to grant it more land, which allows it to hold a greater numbers of farmers, animals and crops. The upgrade may also increase the passive currency income. In order for the simulation to not run into issues, it might be a nice idea to make sure the farm only upgrades once your farm has acquired 20% (choose any % you like though) more than the cost of an upgrade. So, if an upgrade costs $1000, it will automatically upgrade at $1200 so the farm still has $200.
- Up to 6 farmers may start on a single farm with more farmers being hired every few cycles (this is your choice). Once a farm reaches its capacity of 10 farmers, then a new farm must be created by 3 of those farmers.
- Farmers can have affinities for things such as (these are some examples to give you ideas):
  - Being better at growing crops
  - Being better at rearing animals
  - Possess a certain money-making skill
- Animals reside on farms; it is up to you to decide the total number of animals that your farm(s) will hold. Think of typical farm animals such a cows and pigs, but you can be creative if you wish.
- Animal product (milk, wool, and so on) is replenished after every 2nd day. For example, if a sheep is sheared for its wool then the farmers must wait 2 cycles (day and night) to collect the wool again.
- Similar to farmers, animals have affinities too (these are some examples to give you ideas):
  - A horse could be especially fast (higher chance for a person to win a race)
  - A cow may be larger than usual and produce more milk
  - A sheep may produce more wool
- During night cycles predators come out.
  - Predators could be foxes or wolves that may attack/eat the animals
  - Predators could also be rabbits or some other animal that eats crops
  - Predators could also be moles that damage the soil

- Animals have a chance to be born every 4 cycles (must have at least 2 for the chance to occur). Alternatively, as an example, you could specify in your simulation that you wish to spend a certain percentage of your total currency every X number of cycles to buy more animals.
- Animals live for 14 days unless killed by a predator or they become diseased. If killed by a predator they disappear in that same night, if they become diseased they have a chance to die in the next cycle (day and night), unless treated by a farmer. Animal affinities may help with fighting the disease.
- Animals have a natural life cycle just like in real life. They start as a baby where they cannot produce anything for the first 3 days (but may still be killed by predators and disease), and then after that they may produce until they die.
- Crops are grown on farms and have a chance to become diseased. When this happens they have a chance to wither and die within the next cycle (day and night), unless treated by a farmer. Crop affinities may help with fighting the disease.
- Crops may be harvested 1 time every 3 cycles.
- The farmers on your farm are quite tech savvy, so when animals or crops die (or harvested), an automatic message is sent to their supplier notifying them that they need more stock.
- Animals and crops cost currency to purchase. If your farm purchases animals or crops in bulk then they may be bought at a discount depending on the quantity.

It is all pretty wide open and that is on purpose to give you more options and have you think about things more.

Some HINTS (not requirements):

- You could use decorator pattern to append new affinities.
- To build new farms for farmers you could use the factory pattern.
- The simulation should be tick-based. (Mediator pattern). Each tick something should happen, the farm(s) gain more currency, animals or crops die or are born/-planted, etc.

Extra Credit more specifics (15 points)

In your document include a new section for the Design Pattern, explaining your idea briefly.

This is partly already if you make your Design Pattern work together (7 points).

The rest of the points can be gained for the following:

The main thing is that you can run your code through an executable jar file, which requires a JSON file as input. I want to be able to call 'java -jar asurite_DPextra.jar jsonFile.json' to run your application from command line.

The application should read the JSON file. The JSON file should specify the initial setup, eg. how many farms exist, an initial farmer population and animal/crop population (or both), all equipped with names, etc. (whatever you come up with and is needed for your application). You need to provide at least 2 sample JSON files in your repository.

Now your application runs until either the farm runs out of money or animals/crops, or the farm reaches a certain amount of currency or farm upgrade level.

Submission 1 (15 points)

For submission 1 I want you to have your basic setup done. Private repo, simple Main (can just be a hello world) that can be run through Gradle, Travis CI included, Checkstyle, Spotbugs, basic test setup. So basically everything but the real implementation should be done.

I also want so have a short summary in the Readme.md on GitHub which explains your rough idea for the Design Patterns (you can still change your mind later, but I want to see that you already started planning things).

Submit your link to the private repo (ser316asu added as collaborator). No PDF is needed at this point.

This submission has 15 of the overall points (1. - 3pts, 3. - 2pts, 4. 3pts, 5. 3pts, 6. 2pts, 8. 2pts)

Submission 2 (85 point)

You need to submit both design patterns and all that is asked above

1. **A link** to your GitHub repository (yes here and in PDF)
2. **A link** to your Travis CI repository (yes here and in PDF)
3. **A link** to your screencast (yes here and in PDF)
4. **A PDF document** explaining everything that is asked above in the general assignment. Submit the PDF directly on Canvas.