Exam – ITP2200 – Introduction to Software Testing

April 9, 2021

Create a system for managing diets. A diet has several properties such as the duration and the allowed food, which belong to the abstract class Diet. There can be several types of diets, namely:

- VeganDiet: contains only food that does not come from animals (e.g., meat or eggs).
- FlexitarianDiet: close to a vegetarian diet, but meat can be consumed from time to time, to get some proteins.
- HypercaloricDiet: the purpose is to ingest as many calories as possible. Intended mostly for athletes.
- LowCarbDiet: the purpose is to reduce the ingestion of carbs. Intended mostly to reduce weight.

Each of these diets have their own properties, for instance, the preferred meat for the FlexitarianDiet.

A diet is composed of Food. Each Food object can be of four different types, namely: “Recipe”, “Protein”, “Carb”, or “Fat”.

A person is characterized by a weight, a favorite food, and a number of allergies (there could be none). Based on these attributes, a person could be following several diets, but several restrictions apply, as detailed below.

1. Requirements

1. Diets have several restrictions:
    a. If a diet contains any non-vegan food, it is considered not vegan (i.e., isVegan = false).
    b. If a diet contains only vegan food, it is considered vegan, even if it is not a VeganDiet (e.g., it could be a LowCarbDiet).
    c. A VeganDiet cannot contain non-vegan food.
    d. The preferred meat in a FlexitarianDiet MUST be non-vegan food of protein type.
    e. The maximum carb-type foods that can be included in a LowCarbDiet is two.
2. A person can be following any given diet, except in the following cases:
    a. If their favorite food is non-vegan, they cannot follow a VeganDiet.
    b. They cannot follow a diet if they are allergic to 50% or more of the food allowed by the diet.
    c. If they weigh less than the limit set by the VeganDiet or the LowCarbDiet, they cannot be following these diets (for health reasons).
    d. If they weigh more than the limit set by the HypercaloricDiet, they cannot be following this diet (for health reasons).
3. The Diet class should implement the following methods:
    a. Write the duration of a diet in terms of years, months and days, e.g., “This VeganDiet lasts for 2 years, 3 months and 5 days”.
    b. Write the allowed food, e.g., “The following food is allowed in this FlexitarianDiet: Salad, Soup, Apple, Strawberry, Salmon”.
4. The DietManager class should implement methods for the following purposes:

a. Given a Person and a Diet, return true if they are compatible, false otherwise.
b. Given a Person and a list of Food, create a random HypercaloricDiet with the following attributes:
    i. daysDuration: random number between 1 and 100.
    ii. purpose: “Random diet”.
    iii. allowedFood: all the Food from the list that the person is not allergic to.
    iv. isVegan: false if there is some non-vegan Food, true otherwise.
    v. maxWeightKg: random number between Person.weight and Person.weight + 20.
    vi. minCaloriesPerDay: random number between 2000 and 4000.

2. System design

Figure 1 contains an overall system design to get you started. Note that you can make changes and adjustments as needed, provided you can argue for how those changes were made (how did the team decide, what was the process), why those changes were made (provide a convincing reasoning for why your changes improve the system), and discuss the potential impact of such changes (would the changes make the system easier to use, are there any assumptions that you are making that need to hold in future as well, etc.). Also note that you will need to implement more methods tan those depicted in the diagram (e.g., setters and getters for the Person class).

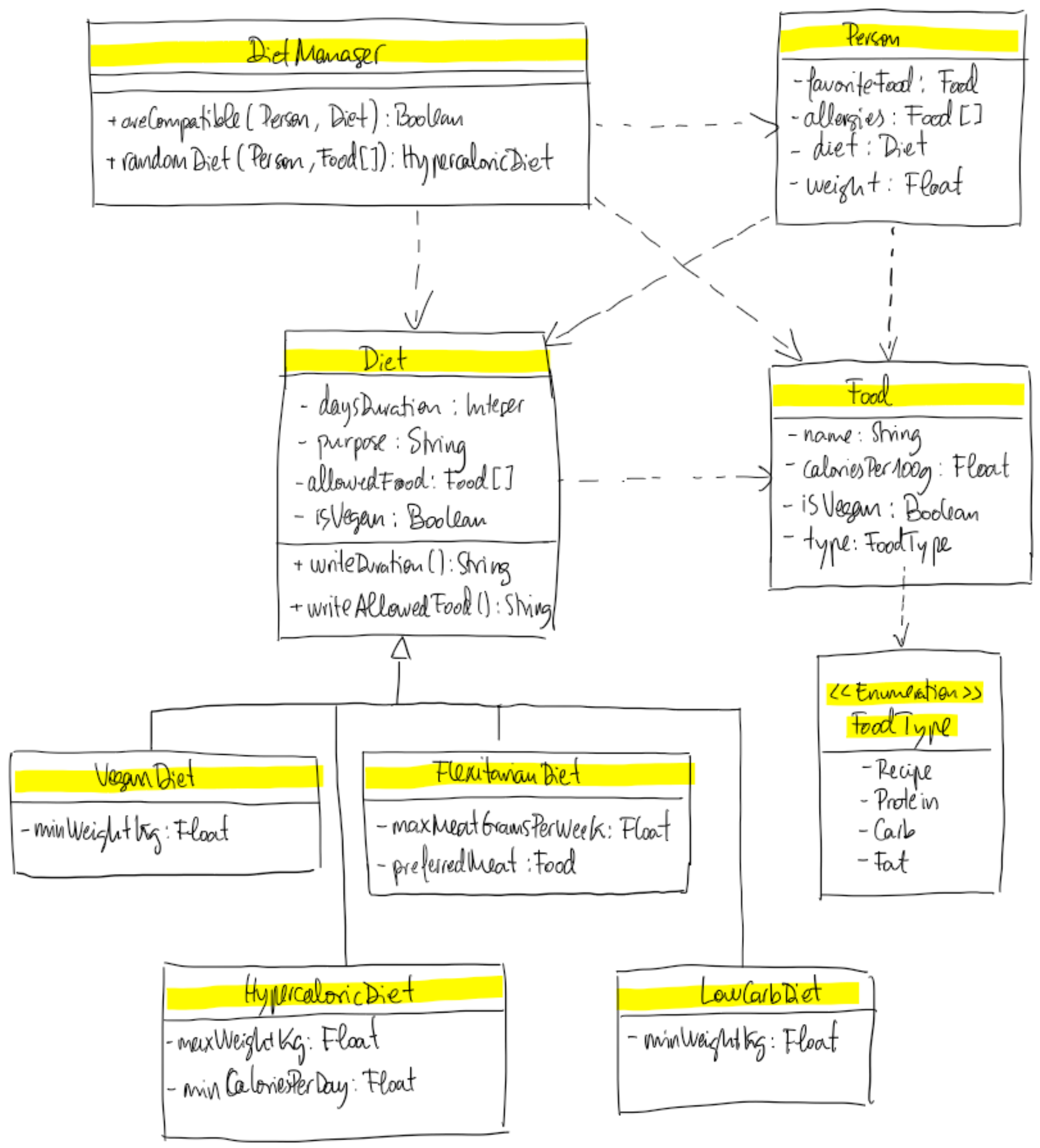

Figure 1. Overall system design.

3. Tasks

1. Write a set of test criteria that evaluate if the requirements are met or not. If you feel that the requirements are not clear or specific enough, motivate why and make a justified assumption for your plan. Include this information in the test plan to be created (see deliverables in the next section).
2. Write a test plan for the system, and update it as needed.
3. Implement the System Under Test (SUT) and the tests. Make sure to include unit tests, integration tests, acceptance tests. You can use TDD or any other techniques, but try to write the tests (especially unit tests) either before or as you are writing the code (rather than long after).
4. Pick a method (one that has interesting behavior) and draw its graph. Then discuss how the different tests you have written follow different paths through that graph (nodes and edges).

4. Deliverables

The final deliverables will be:

1. (Group) The test plan (updated during development, of course). This includes the test criteria, their link to the requirements, and refers the reader to the relevant test classes and methods.
2. (Group) The SUT and test code (archived in a single .zip file). The system should compile and run. You should have a Readme file (readme.md) that describes how to compile, run, and what to expect from your system. All the tests should pass, coverage should be above 50%.
3. (Group) A document (no more than 10 pages), discussing your reflections on the project. An overview of the project progress, the development work (how were roles divided?, how did the team manage?, how much did each member contribute and how?), and the testing you have conducted. A discussion on what tests you could include further. A discussion on any obstacles you have encountered and how you overcame them.
4. (Group) A document including the method picked for analysis, its graph and the discussion of the paths followed by different test cases (task 4 explained above).
5. (Individual) Each team member should have a short document (2-page maximum) discussing their personal contribution and reflections on how the team's approach to collaboration in testing could be improved.

5. Clarifying notes

This is a pass/fail course, therefore no marks will be assigned to the project. However, do consider the following requirements, which must be met to pass the course:

- All deliverables MUST be submitted. The lack of any of the 5 deliverables explained above will mean a failing grade.
- The SUT MUST implement the four requirements explained in Section 1. The lack of any of the requirements will mean a failing grade.
- The tests MUST reach 50% coverage according to IntelliJ. A coverage below 50% will mean a failing grade.

Despite not being marked, the deliverables will be assigned the following percentages. Reaching 50% or more of the overall score means a passing grade:

1. **Test plan: 20%**.
2. **SUT and tests: 40%**. Note: it is possible to get an extra 10% (i.e., more chances to pass if the other deliverables get bad scores) by implementing additional features (e.g., a new Diet, a method for recommending diets for a given person, etc.). If new features are implemented, this must be explicitly stated and explained in the group reflection document (deliverable 4), in a section called "Additional features".
3. **Group reflection document: 20%**.
4. **Task 4 document: 10%**.
5. **Individual reflection document: 10%**.

Deadline: April 30, 2021.