IS1220 - Final Project
Enjoy Your Meal System

CentraleSupélec

Hand-out: February 24, 2016
Due Part 1: March 20, 2016
Due Part 2: April 10, 2016
Programming Language: Java

1 Overview

The goal of this project is to develop a software for a restaurant chain: **Enjoy Your Meal System** (EYMS). This chain provides the possibility to order take-away meals using a system that is available at the entrance of each restaurant. This system provides functionalities, to the clients (that have to be able to order a meal using the interface) and to the manager of the restaurant (that has to be able to set-up the meals).

The project consists of two parts:

Part 1: **EYMS core**, is concerned with the development of basic functionalities for the EYMS system (hence designing of the Java infrastructure for representing/dealing-with the system);

Part 2: **EYMS interface**, is concerned with the development of an interface which allows the actor(s) to interact with the EYMS.

The requirements for both parts of the project are given in Section 2.

2 Project requirements

2.1 Part 1: EYMS core

The restaurant offers the possibility to order meals by accessing a system available at the main entrance of the restaurant.

- Each restaurant offers a set of predefined meals, that can be selected as default choices by all clients. Each meal is characterized by a set of ingredients. Each ingredient has a name and a quantity. The restaurant offers also the possibility to personalize meals by adding or removing ingredients from any choice.
- The clients are characterized by a first-name, a last-name and a username. The system stores also informations about their address, their general contact information (phone number, email, ...), and their favorite meals. A client has also the possibility to specify his agreement to receive general and/or personalized (birthday special offers, new meals alert, ...) promotions. Of course the system has to store how to contact a client that agrees to receive promotions (email, phone, ...). A client can also have a fidelity card. There are many types of fidelity cards. For example the **basic fidelity card** will allow to access to special offers, the **point fidelity card** will also to gain points, etc.

The requirements are described according to the following use case scenarios of EYMS:

Startup scenario

1. the system loads all the users
2. the system loads all the available meals
3. the system sends alerts to the users

Register a user

1. a client start using the system because she wants to register
2. the client inserts his first-name, his last-name, his username
3. the user starts inserting a contact info with the type and the value
    - the user repeats step 3 since he ends to inserts his contact info

4. the user sets the agreement about the special offer contact (by default it is no)
5. the user selects the contact to be used to send the offers (by default it is the e-mail if exists)
6. the client specify to save the account

Login user

1. a user wants to login
2. the user inserts username and password
3. the system handles the login and presents to the user the available operations according to his role

Ordering a meal

1. a client start using the system because she wants to order a meal
2. the client inserts his credentials (username and password)
3. the system recognizes the client and proposes the available meals
4. the client selects the meal she wants to order and specify the quantity (for 1 people, 2 peoples, etc.)

   4.1 the client insert a personalization for one ingredient for the meal
   - the client repeats step [4.1] till she has finished the personalization
   - the client saves the meal to order

   • the client repeats step 4 till she has no more meals to add to her order
5. the client selects the end of her order
6. the system shows the summary of the ordered meals and the total price of the order taking into account the pricing rules

Inserting a meal

1. the chef of the restaurant start using the system because she wants to insert a new meal

2. the chef inserts her credentials (username and password)
3. the system recognizes the chef and shows the available meals
4. the chef selects the insert new meal operations
5. the chef inserts the name of the new meal
6. the chef inserts the ingredients of the meal and their quantity
7. the chef inserts the price of the meal
8. the chef saves the new created meal

Notice that the modify meal operation does not modify the meal offered by the restaurant but just the order of the user.

Adding a price special offer

1. the chef starts using the system and inserts her credentials
2. the system shows all the available meals
3. the chef selects the meal and specifies the new price
4. the system shows the list of meals in a special offer
5. the system notifies the users (that agreed to be notified of special offers) about the new offer

Removing a price special offer

1. the chef starts using the system and inserts her credentials
2. the system shows all the available meals
3. the chef selects a meal in special offer and selects the remove special offer operation

Sending a birthday offer

1. the system automatically checks the clients that celebrate their birthday at the beginning of the day[1]
2. the system send the birthday offer to the clients that agreed on receiving it

Hint. Which design pattern helps you solving this problem?

The pricing policy depends on the kind of fidelity card that a user owns. The EYMS handles different types of cards, including the following three types: **basic fidelity card**, **point fidelity card** and **lottery fidelity card**. The final price is then calculated according to the following rules:

1. **basic fidelity card** is the card given by default, at registration, to any user. This card simply allows to access to special offers that are provided by the restaurant
2. **point fidelity card**. A client can select to have this fidelity card. Instead of having the special offer she will gain a point for each 10 euros spent in the restaurant. Once she will reach 100 points she will receive a 10% discount on the next order.
3. **lottery fidelity card**. A member that has this card will not access to any offer nor gain any points but will have a certain probability to gain her meal for free each day[2]

Hint. Which design pattern helps you solving this issue?.

Storing of orders: The EYMS should allow for storing all the shipped orders. Given those orders the members of the restaurant can analyze the most popular meals and the most popular personalizations. The system can show the meals ordered according to the following criteria:

1. **As it is**: the meals ordered as they are
2. **Mostly modified**: the meals that are ordered mostly in a modified version
3. **Just on sale**: the meals that are ordered just when there is a special offer

Remark. The implementation of the storing strategies within the EYMS should obey the OPEN-CLOSED principle (i.e. a flexible solution should be implemented).

[1] In your first implementation you can do this at the start-up of the system.
[2] In the initial implementation you can do this at the startup of the system.

2.2 Part 2: EYMS user interface

The part 2 of the project is about realizing a user interface for the EYMS-app. The user interface consists of two parts: the command-line user interface (CLUI), which is mandatory, and the the graphical user interface (GUI), which is optional (but will gain extra points, which can re-enforce points lost elsewhere).

2.2.1 EYMS Command Line User Interface

The command line interpreter provides the user with a (linux-style) terminal like environment to enter commands to interact with the EYMS core. In particular command-line interpreter should feature the following list of commands:

- `login <username, password>` : to perform the login
- `createMeal <mealName, price>` : to create a mail with a given name
- `addIngredient <ingredientName, quantity>` : to add an ingredient to the current meal
- `currentMeal <>` : to show the name of the under construction meal
- `saveMeal <>` : to save the current meal
- `selectMeal <mealName, quantity>` : to add quantity meals of a given name to an order
- `personalizeMeal <mealName,ingredient,quantity>` : to add/remove an ingredient from a meal
- `putInSpecialOffer <mealName, price>` : to add a meal in a special offer
- `removeFromSpecialOffer <mealName>` : to reset a special offer
- `listIngredients <mealName>` : shows all the ingredients of a meal
- `saveOrder <>` : to save the current order
- `insertOffer <mealName,newPrice>` : to insert a meal in a special offer policy
- `registerClient <firstName, lastName, username, password>` : to add a user to the system
- `addContactInfo<contactInfo>` to add a contact info to the current user

- `associateCard <userName,cardType>` to associate a fidelity card to a user
- `associateAgreement <username,agreement>` to associate an agreement to a user
- `insertChef <firstName, lastName, username, password>` : to add a chef to the system
- `notifyAd <message, mealName, specialPrice>` send a message to alert the users about an offer
- `notifyBirthday <>` send a special offer to the users that celebrate their birthday
- `showMeal <orderingCriteria>` to see the list of the meals ordered according to a certain criteria

It should be possible to write those commands on the CLUI and to run the commands in an interactive way (reading/parsing the commands from a textual file is not sufficient).
Error messages and CLUI. The CLUI must handle all possible types of errors, i.e. syntax errors while typing in a command, and misuse errors, like for example trying to order a meal which is not contained in the menu, or modifying a meal that is not in the menu, etc.

2.2.2 EYMS Graphical User Interface

The goal of this part is to realize an interactive Graphical User Interface (GUI) for the EYMS core. The GUI must support all operations implemented in Part 1. Navigation should be possible using both mouse and keyboard. For this part of the project you only have to support a single user; multi-user support is not required. You are encouraged to design a nice and user-friendly interface. An interactive GUI must provide for example the possibility to select a menu, to visualize this menu (using a click on the name of the menu, or on a button, ...), etc. Bonus points are given for a responsive GUI that does not stall during long-running operations, like search of meals in a menu.

EYMS GUI requirements

1. The browser should support all operations from Part 1 (EYMS core). For example, users should be able to create a new meal, add ingredients, etc.) all of which without using console commands.

2. The browser should support the login of the user showing the available operations only to a certain category of users (a client cannot insert a new meal).

3. The browser should support mouse navigation. The required operations are the same as in requirement.

2.3 Hints

1. Develop a set of JUnit tests prior to the development, try to work using the Test-Driven-Development (TDD) approach.

2. for GUI related problems it might be useful to have a look to online documentation for Java Swing, for example `http://docs.oracle.com/javase/tutorial/uiswing/dnd/index.html` and in particularly for GUI reactivity `http://docs.oracle.com/javase/tutorial/uiswing/concurrency/index.html`

3 Tests JUnit

All projects must come with a JUnit test for testing each functionality (each command of the CLUI) in isolation. The project must then contain a summary *test scenario*, called **test1**. We suppose that a waiter (called "Bob Red", with username "bobred" and with password "123456" is already registered in the system and is correctly handled by the system at the startup). You must handle the following steps:

- execute the `login<‘‘bobred’’, ‘‘123456’’>` operation for *"Bob Red"*
- execute the `createMeal<‘‘Raclette’’>` to create a meal named *"Raclette"*
- execute `addIngredient <ingredientName, quantity>` to add 3 ingredients (cheese 90g, ham 40g, potatoes 50g) to the "Raclette"
- verify the `currentMeal<>` operation
- save your meal using `saveMeal<>`
- add 20 meals to the restaurant specifying at least one ingredient for each. Among the added meals at least 3 should share the ingredient "sugar" with different quantity

- execute the `listIngredients<‘‘Raclette’’>` CLUI command to display the content of the *“Raclette”* meal
- execute the `registerClient <firstName, lastName, username, password>` to register a cliend *“Mario Rossi”* with username *“Mario”* and password *“345678”*
- add the email address and the phone number for the client performing the `addContactInfo<contactInfo>` operation
- select a card for the client using the `associateCard<userName, cardType>` operation
- associate an agreement level to each user using `associateAgreement <username, agreement>` operation
- execute the `login<‘‘mario’’, ‘‘345678’’>` operation for *“Mario Rossi”*
- check the ingredients of the *“Raclette”* using the `listIngredients < ‘‘Raclette’’>`
- order one *Raclette* using `selectMeal<mealName, quantity>` operation
- modify the *Raclette* meal using the command `personalizeMeal <mealName, ingredient, quantity>` to add 10g of cheese to the raclette. Handle the case in which the quantity of an ingredient is modified to 0
- order two standard *Raclette* using `selectMeal<mealName, quantity>` operation
- save the order executing the command `saveOrder<>`
- send a message to a user that has his birthday in the same day using the `notifyBirthday <>` operation (for this operation it is not needed that the email is sent, just that the system correctly recognizes the set of user to whom send a message)
- execute the `login<‘‘bobred’’, ‘‘123456’’>` operation for *“Bob Red”*
- put the meal *“Raclette”* in special offer using the operation `putInSpecialOffer <mealName, price>`
- the system sends a special message using the `notifyAd <message, mealName, specialPrice>` operation

Only the parts documented and checked using JUnit tests will be considered for the evaluation of the project.
The **test1** test-scenario must be stored in a file called "eval_1.txt" inside the directory "eval" directory (see details in Section 4.2)

REMARK: all tests provided with the project must be clearly described in the project report in a dedicate *JUnit Test* section. For each test unit/scenario students must describe: what functionality the test is covering, how it is designed, what are the expected outcomes, and if a test fails a discussion on why it fails and what shall be done to fix it.

4 Deadlines and submitting instructions

All project work must be done in teams of 2 people. The project itself is divided in 2 parts, each with its own deadline:

- **Part 1, the EYMS core**: due **March 20, 2016** (i.e. $\sim$ 4 weeks work)
- **Part 2, the EYMS user-interface**: due **April 10, 2016** (i.e., $\sim$ 2 weeks work) which can consist in either:
  - a command-line shell: allowing the user to interact with the EYMS core through a number of pre-defined commands
  - a graphical user interface: allowing the user to interact with the EYMS core through a GUI

At each deadline you must hand in:

- an Eclipse project containing the code for your implementation (see details below for how to name the project and what the project should contain)
- a written report on the design and implementation of the corresponding part

Note that the proposed project structure allows for incremental development (you can first work out part 1, and then part 2 which will be based on part 1).

4.1 Project naming

You must hand over your work for both parts of the projects through Claroline (Assignments section) in the form of an Eclipse project (exported into a .zip file). The Eclipse projects you submit must be named as follows:

- `Group`N`_Project_IS1220_part1_student1Name_student2Name` (the project covering part 1), exported into file `Group`N`_Project_IS1220_part1_student1Name_student2Name.zip`
- `Group`N`_Project_IS1220_part2_student1Name_student2Name` (the project covering part 2), exported into file `Group`N`_Project_IS1220_part2_student1Name_student2Name.zip`

thus, if Group1 is formed by students Alan Turing and John Von Neumann, they will have to create, on Eclipse, a project called `Group1_IS1220_Project_part1_Turing_VonNeumann`, which they will export into a file named, `Group1_IS1220_Project_part1_Turing_VonNeumann.zip`.

4.2 Eclipse project content

Each Eclipse project should contain all relevant files and directories, that is:

- .java files logically arranged in dedicated *packages*
- a "test" package containing all relevant junit tests
- a "doc" folder containing the *javadoc* generated documentation for the entire project
- a "model" folder containing the papyrus UML class diagram for the project
- an "eval" folder containing the following:
  - the generation of javadoc is essential for the evaluation of the project. Think about writing javadoc comments as soon as you start writing your code: at the end is not useful for you and it is time consuming.
  - The UML class diagram must be attached to the project as an image (we encourage you to use papyrus but you can design and produce this file using any tool you like).
  - **MANDATORY**: at least a file "eval_N.txt" which contains a sequence of CLUI commands that allows to test the main functionalities of the EYMS-app following the test cases described in Section 3

IMPORTANT REMARK: it is the students responsibility to ensure that the project is correctly named as described above and that it is correctly exported into a .zip archive that correctly works once is imported

back into Eclipse. A PROJECT THAT IS NOT PROPERLY NAME OR THAT CANNOT BE STRAIGHTFORWARDLY IMPORTED IN ECLIPSE WILL NOT BE EVALUATED (HINT: do verify that export and re-import works correctly for your project work before submitting it).

5 Writing the report (team work)

. You also have to write a final report file describing your solution. The report **must** include a detailed description of your project and must comprehend the following points:

- main characteristics
- design decisions
- used design patterns
- advantages/limitations of your choices
- how you organized the work (who did what)
- etc.

The quality of the report is an important aspect of the project's mark, thus it is warmly recommended to write a quality report. The report should also describe how the work has been divided into task, and how the various tasks have been allocated between the two members of a group (e.g. task1 of EYMS-core → responsible: Arnault, task2 of EYMS-core →responsible: Vladimir, etc.). Also the writing of the report should be fairly split between group's members with each member taking care of writing about the tasks he/she is responsible for. In a good report there is no code listing but some code can be inserted in order to comment special algorithms or issues (do not abuse of this). You can find a reference guide for writing your report at: `http://www.cs.ucl.ac.uk/students/msc_cgvi/projects/project_report_structure/`, part 3.

6 Project grading

The project is graded on a total of 100 points for mandatory features + 20 bonus points (EYMS GUI) (bonus points will complement points lost elsewhere). The guidelines of marks breakdown is given below:

- EYMS Core functionalities (max 40 point)
- EYMS CLUI functionalities (max 20 points)
- JUnit tests (max 15 points): EYMS core (9pt), EYMS CLUI (6pt)
- UML (max 10 points): EYMS core (8pt), EYMS CLUI (2pt)
- Final report (between -15 and +15 points depending on quality)
- EYMS GUI (max 20 points, bonus)

Each part of the solution (i.e. EYMS CORE, EYMS CLUI, and also EYMS GUI) will be evaluated according to two basic criteria:

- requirements coverage: how much the code meets the given project requirements (described in Section 2)
- Code quality: how much the code meets the basic principles of OOP and software design seen throughout the course (i.e. object oriented design, separation of concerns, code flexibility, application of design patterns, etc.)
- the quality of the report describing each part of the project

The grade will be determined based on the project as final implementation and the final report as submitted by April 10, 2016. However, each team has to provide an implementation and a report summarizing the current design and project status at each intermediate deadline (March 20, 2016). The implementation at the intermediate deadlines has to be runnable (they must compile and run).

IMPORTANT: if a group does not submit part 1 (but only part 2) some points will be deducted from the final mark.

Remark: the above grading scheme is meant to give an idea of the relative importance of each part of the project. It will be used as a guideline throughout the marking but it does not constitute an obligation the marker must stick to. The marker has the right to adapt the marking criteria in any way he/she feels like it is more convenient in order to account for specific aspects encountered while marking a particular solution.

7 General Remarks

While working out your solution be aware of the following relevant points:

- Design your application in a modular way to support separation of concerns. For example, EYMS core should not depend on the EYMS command-line user interpreter.
- The system should be robust with respect to incorrect user input. For example, when a user tries to import files from a nonexistent directory, the EYMS should not crash.
- **application of design patterns**: flexible design solutions must be applied whenever appropriate, and missing to apply them will affect the evaluation of a project (i.e., a working solution which doesn't not employ patterns will get points deducted). Thus, for example, whenever appropriate decoupling approaches (e.g. visitor pattern) for the implementation of specific functionalities that concern EYMS must be applied.

8 Questions

Got a question? Ask away by email:

Paolo Ballarini:	paolo.ballarini@ecp.fr
Francesca Bugiotti	francesca.bugiotti@supelec.fr
Arnault Lapitre:	arnault.lapitre@gmail.com
Vladimir Paun:	paun@ensta-paristech.fr

or post it on the Forum page on Claroline.