Office Calendar

Object-Oriented Programming
2nd Project, version 1.1, 2024-05-21
Deadline until 5pm on May 31, 2024

Important Notes

- This project should be developed using Java features already covered in the OOP course.

1 Introduction

This document describes the 2nd OOP course project of the 1st year of the degree in Computer Science. It also describes the work methodology students must follow to develop the project.

The project must be carried out in groups of 2 students. Students can and should clarify any doubts with the teaching team, as well as discuss with their colleagues, always respecting the Code of Ethics principles available on the course page.

This project's goal is to program, in Java, "Office Calendar" described in this document.

2 Application Description

The goal of this project is to develop an application for managing a given company's cloud-based calendar system, similar to Google Calendar.

Each company user has access to a calendar where they can schedule events. We assume that the company events are accessible through a website. Users can invite other registered members to participate in these events.

Events are described later, but they include details such as its promoter (account name), date, time, priority level, and the meeting topics. Note that multiple meetings can overlap in time. There are three types of accounts: staff, manager, and (external) guest accounts. Staff can create and invite guests to events, while guests can only accept or decline invitations. Managers have additional capabilities. Managers are the only ones who have the authority to schedule high-priority events. On the other hand, staff members can only schedule events with medium priority. Additionally, staff members' accounts are programmed to automatically accept high-priority event invitations, even if they conflict with their existing commitments. However, managers have the right to decline such invitations. External guests are allowed to respond to event invitations, but they cannot create events nor be compelled to accept invitations.

As stated before an event is defined by its name, timing, priority, promoter's account, and the list of invited users. The guest list specifies each participant's response status—whether they've accepted, declined, or not yet responded. The promoter is automatically listed as a confirmed attendee. Events are uniquely identified by their name and the promoter's account. Lastly, all events have a fixed duration of one hour, simplifying the process of determining scheduling conflicts.

3 Commands

In this section we present all the commands that the system must be able to interpret and execute. In the following examples, we differentiate `text written by the user` from the feedback written by the program in the console. You may assume that the user will make no mistakes when using the program other than those described in this document. In other words, **you only need to take care of the error situations described here, in the exact same order as they are described in this document**.

Commands are case insensitive strings with no blank spaces inside. For example, the `exit` command may be written using any combination of upper and lowercase characters, such as `EXIT`, `exit`, `Exit`, `exIT`, and so on. In the examples provided in this document, the symbol ↵ denotes a change of line.

If the user introduces an unknown command, the program must write in the console the message (Unknown command COMMAND. Type help to see available commands.). For example, the non existing command `someRandomCommand` would have the following effect:

```
someRandomCommand↵
Unknown command SOMERANDOMCOMMAND. Type help to see available commands.↵
```

If there are additional tokens in the line (e.g. a parameter for the command you were trying to write), the program will try to consume them as commands, as well. So, in this example, `someRandom Command` would be interpreted as two unknown commands: `someRandom` and `Command`, leading to two error messages.

```
someRandom Command↵
Unknown command SOMERANDOM. Type help to see available commands.↵
Unknown command COMMAND. Type help to see available commands.↵
```

Several commands have arguments. Unless explicitly stated in this document, you may assume that the user will only write arguments of the correct type, in the correct order. However, some of those arguments may have an incorrect value. For that reason, we need to test each argument exactly by the order specified in this document. Arguments will be denoted **`with this style`**, in their description, for easier identification. Also, for any String arguments in the commands, assume they are case-sensitive. So, "Yet Another Meeting" and "Yet another meeting"' would be two different meeting names, for instance.

3.1 `exit` command

Terminates the execution of the program. This command does not require any arguments. The following scenario illustrates its usage.

```
exit↵
Bye!↵
```

This command always succeeds.

3.2 `help` command

Shows the available commands. This command does not require any arguments. The following scenario illustrates its usage.

```
help↵
Available commands:↵
register - registers a new account↵
accounts - lists all registered accounts↵
create - creates a new event↵
events - lists all events of an account↵
invite - invites an user to an event↵
response - response to an invitation↵
event - shows detailed information of an event↵
topics - shows all events that cover a list of topics↵
help - shows the available commands↵
exit - terminates the execution of the program↵
```

This command always succeeds. When executed, it shows the available commands.

3.3 Command `register`

Registers a new account the system. The command receives as arguments the **`account name`** (the user's email address) and the account type (`staff`, `manager`, or `guest`). The expected behaviour is as follows: If there is no registered account with that **`account name`**, the account is registered and the message (<account name> was registered.) is written as feedback. The following scenario illustrates its usage.

```
register pam.beesly@dunder.mifflin.com staff↵
pam.beesly@dunder.mifflin.com was registered.↵
register michael.scott@dunder.mifflin.com manager↵
michael.scott@dunder.mifflin.com was registered.↵
register bob.vance@vance.refrigeration.com guest↵
bob.vance@vance.refrigeration.com was registered.↵
```

The following errors may occur:

1. The account **`account name`** already exists, the adequate error message is (Account <account name> already exists.).
2. The type of account to be registered is unknown (Unknown account type.).

```
register pam.beesly@dunder.mifflin.com staff↵
pam.beesly@dunder.mifflin.com was registered.↵
register pam.beesly@dunder.mifflin.com manager↵
Account pam.beesly@dunder.mifflin.com already exists.↵
register andy.bernard@dunder.mifflin.com cornell↵
Unknown account type.↵
```

3.4 Command `accounts`

Lists all registered accounts. This command does not receive any arguments and always succeeds. If the application does not have any registered accounts, the output is (No accounts

registered.). Otherwise, it will present a header line (All accounts:) followed by summary information about all the registered accounts, ordered alphabetical ordered by account name, one by line, with the following format (<account name> [<account type>]). The following scenario illustrates its usage.

```
accounts↵
No accounts registered.↵
... a few accounts registered ...
accounts↵
All accounts:↵
andy.bernard@dunder.mifflin.com [staff]↵
angela.martin@dunder.mifflin.com [staff]↵
bob.vance@vance.refrigeration.com [guest]↵
michael.scott@dunder.mifflin.com [manager]↵
pam.beesly@dunder.mifflin.com [staff]↵
```

3.5 Command `create`

Creates an event. The command receives as arguments the **account name**, **the event name**, the **priority** (`mid` or `high`), and the **day** and **time** of the event (text in the format YYYY MM DD HH), and the event **topics** which are words separated by a space (it can be assumed there is a least one topic). The class `LocalDateTime` can be used to keep the date and time of the event. Moreover, it can be assumed that the date and time are always valid.

If successful, the event is created and added to the promoter's account and the message (<event name> is scheduled.) is written as feedback. As said before, the event promoter is automatically added as a confirmed attendee. The following scenario illustrates its usage.

```
accounts↵
All accounts:↵
andy.bernard@dunder.mifflin.com [staff]↵
angela.martin@dunder.mifflin.com [staff]↵
bob.vance@vance.refrigeration.com [guest]↵
michael.scott@dunder.mifflin.com [manager]↵
pam.beesly@dunder.mifflin.com [staff]↵
create angela.martin@dunder.mifflin.com↵
The launch party↵
mid 2024 05 31 17↵
website launch scranton↵
The launch party is scheduled.↵
create michael.scott@dunder.mifflin.com↵
Integration celebration↵
high 2024 04 28 14↵
merger stamford scranton↵
Integration celebration is scheduled.↵
```

The following errors may occur:

1. The account with **account name** does not exist (Account <account name> does not exist.).

2. The event **priority** is unknown (Unknown priority type.).
3. Guest account **account name** cannot create new events (Guest account <account name> cannot create events.).
4. The account **account name** is not allowed to create high priority events (Account <account name> cannot create high priority events.).
5. An event with **event name** already exists in **account name** (<event name> already exists in account <account name>.).
6. The promoter **account name** already has accepted another event at the same time (Account <account name> is busy.).

```
create grogu@mandalorian.com↵
Din Grogu celebration↵
mid 2025 07 12 10↵
mandalore jedi↵
Account grogu@mandalorian.com does not exist.↵
create toby.flenderson@dunder.mifflin.com↵
Yet another health and safety meeting↵
low 2024 12 15 12↵
health safety↵
Unknown priority type.↵
create angela.martin@dunder.mifflin.com↵
Party planning committee meeting↵
high 2024 05 12 09↵
celebration birthday↵
Account angela.martin@dunder.mifflin.com cannot create high priority events.↵
create michael.scott@dunder.mifflin.com↵
Integration celebration↵
mid 2025 12 10 15↵
merger stamford scranton↵
Integration celebration already exists in account michael.scott@dunder.mifflin.com.↵
create bob.vance@vance.refrigeration.com↵
Air conditioners sale↵
mid 2024 06 02 10↵
ac sale↵
Guest account bob.vance@vance.refrigeration.com cannot create events.↵
create michael.scott@dunder.mifflin.com↵
Koy pond↵
mid 2024 04 28 14↵
fall awkward video↵
Account michael.scott@dunder.mifflin.com is busy.↵
```

3.6 Command **events**

Lists all events of an account. This list should include both the events promoted by the user of the account and the events to which the user has been invited. The command receives as arguments the **account name**. If the user has not promoted or been invited to any event,

the output is (Account <account name> has no events.) Otherwise, it will present a header line (Account <account name> events:) followed by summary information about all the events, presented one by line in the order the events were registered into the user's account. The information shown for each event should follow the following format (<event name> status [invited <number of invitations>] [accepted <number of accepted invitations>] [rejected <number of rejected invitations>] [unanswered <number of unanswered invitations>]). The following scenario illustrates its usage.

```
accounts↵
All accounts:↵
andy.bernard@dunder.mifflin.com [staff]↵
angela.martin@dunder.mifflin.com [staff]↵
bob.vance@vance.refrigeration.com [guest]↵
michael.scott@dunder.mifflin.com [manager]↵
create angela.martin@dunder.mifflin.com↵
The launch party↵
mid 2024 05 31 17↵
website launch scranton↵
The launch party is scheduled.↵
events andy.bernard@dunder.mifflin.com↵
Account andy.bernard@dunder.mifflin.com has no events.↵
events angela.martin@dunder.mifflin.com↵
Account angela.martin@dunder.mifflin.com events:↵
The launch party status [invited 1] [accepted 1] [rejected 0] [unanswered 0]↵
... after several commands are executed ...↵
events angela.martin@dunder.mifflin.com↵
Account angela.martin@dunder.mifflin.com events:↵
The launch party status [invited 10] [accepted 5] [rejected 3] [unanswered 2]↵
Secretary's day status [invited 14] [accepted 4] [rejected 1] [unanswered 9]↵
```

The following error may occur:

1. The account with **account name** does not exist (Account <account name> does not exist.).

```
All accounts:↵
andy.bernard@dunder.mifflin.com [staff]↵
angela.martin@dunder.mifflin.com [staff]↵
michael.scott@dunder.mifflin.com [manager]↵
events grogu@mandalorian.com↵
Account grogu@mandalorian.com does not exist.↵
```

3.7 Command `invite`

Invites an user to an event. The command receives as arguments the **invitee account name**, the **account name** of the event promoter, and the **event name**.

In case of success there are several possible feedback messages.

- If the invitation concerns a staff user and a high priority event then (<guest name> accepted the invitation.) is written as feedback followed by two alternative outputs.

  - If accepting this invitation creates a time conflict with other events in which the (staff) invitee is involved, those events are rejected even if one of them was already accepted. In this case the output message is followed by all rejected events, sorted by the order the invitations were received, one by line, with the following format (<other event name> promoted by <promoter account> was rejected).
  - If accepting this invitation causes the invitee to reject an event that they promoted, then the event promoted by the staff user has to be removed, as they cannot attend the event, then the output message is followed by information on the removed event (<other event name> promoted by <invitee account name> was removed.).
  - Note that if the staff invitee has previously accepted a high priority event, then the current, more recent, high priority invitation is rejected and the message (<invitee name> is already attending another event.) is written as feedback (error situation item 4).

- Otherwise, the message (<guest name> was invited.) is presented.

Note that guest users are never obliged to accept invitations.

The following scenario illustrates its usage.

```
accounts↵
All accounts:↵
andy.bernard@dunder.mifflin.com [staff]↵
angela.martin@dunder.mifflin.com [staff]↵
bob.vance@vance.refrigeration.com [guest]↵
michael.scott@dunder.mifflin.com [manager]↵
pam.beesly@dunder.mifflin.com [staff]↵
create angela.martin@dunder.mifflin.com↵
The launch party↵
mid 2024 05 31 17↵
website launch scranton↵
The launch party is scheduled.↵
create michael.scott@dunder.mifflin.com↵
Integration celebration↵
high 2024 04 28 14↵
merger stamford scranton↵
Integration celebration is scheduled.↵
invite pam.beesly@dunder.mifflin.com↵
angela.martin@dunder.mifflin.com The launch party↵
pam.beesly@dunder.mifflin.com was invited.↵
invite pam.beesly@dunder.mifflin.com↵
michael.scott@dunder.mifflin.com Integration celebration↵
pam.beesly@dunder.mifflin.com accepted the invitation.↵
register david.wallace@dunder.mifflin.com manager↵
david.wallace@dunder.mifflin.com was registered.↵
create david.wallace@dunder.mifflin.com↵
Plenary meeting↵
high 2024 05 31 17↵
merger↵
Plenary meeting is scheduled.↵
```

```
invite andy.bernard@dunder.mifflin.com↵
angela.martin@dunder.mifflin.com The launch party↵
andy.bernard@dunder.mifflin.com was invited.↵
invite andy.bernard@dunder.mifflin.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
andy.bernard@dunder.mifflin.com accepted the invitation.↵
The launch party promoted angela.martin@dunder.mifflin.com was rejected.↵
invite michael.scott@dunder.mifflin.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
michael.scott@dunder.mifflin.com was invited.↵
invite bob.vance@vance.refrigeration.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
bob.vance@vance.refrigeration.com was invited.↵
invite angela.martin@dunder.mifflin.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
angela.martin@dunder.mifflin.com accepted the invitation.↵
The launch party promoted by angela.martin@dunder.mifflin.com was removed.↵
```

The following errors may occur:

1. One or both of the accounts do not exist (Account <non-existing account name> does not exist.), where the account name reported is the first found not to be registered.
2. The event with **event name** does not exist in the **account name** (<event name> does not exist in account <account name>.).
3. The **account invitee** has already been invited to the event with **event name** (<account invitee> was already invited.).
4. The **account invitee** has already accepted the invitation for another event at the same time (<account invitee> is already attending another event.).

```
... assume the above commands are executed ...
invite grogu@mandalorian.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
Account grogu@mandalorian.com does not exist.↵
invite bob.vance@vance.refrigeration.com↵
grogu@mandalorian.com Some meeting↵
Account grogu@mandalorian.com does not exist.↵
invite gareth.keenan@wernham.hogg.co.uk↵
grogu@mandalorian.com Some meeting↵
Account gareth.keenan@wernham.hogg.co.uk does not exist.↵
invite david.wallace@dunder.mifflin.com↵
angela.martin@dunder.mifflin.com Integration celebration↵
Integration celebration does not exist in account angela.martin@dunder.mifflin.com.↵
invite bob.vance@vance.refrigeration.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
bob.vance@vance.refrigeration.com was already invited.↵
```

```
invite bob.vance@vance.refrigeration.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
bob.vance@vance.refrigeration.com was already invited.↵
create david.wallace@dunder.mifflin.com↵
Alternative integration celebration↵
high 2024 04 28 14↵
merger stamford scranton↵
Alternative integration celebration is scheduled.↵
invite michael.scott@dunder.mifflin.com↵
david.wallace@dunder.mifflin.com Alternative integration celebration↵
michael.scott@dunder.mifflin.com is already attending another event.↵
invite pam.beesly@dunder.mifflin.com↵
david.wallace@dunder.mifflin.com Alternative integration celebration↵
pam.beesly@dunder.mifflin.com is already attending another event.↵
```

3.8 Command `response`

An user responds to an event invitation. The command receives as arguments the **`invitee account name`**, the **`account name`** of the promoter, the **`event name`**, and the **`response`** to the invitation which can be either `accept` or `reject`. When an user accepts an invitation, all other invitations at the same time are automatically rejected. If successful, the invitee's status is updated to indicate whether they accepted or rejected to participate in the event (Account <invitee account name> has replied <response> to the invitation.). If the invitation was accepted, the output should present all rejected events, sorted by the order the invitations were received, one by line, with the following format (<other event name> promoted by <promoter account> was rejected).

```
accounts↵
All accounts:↵
andy.bernard@dunder.mifflin.com [staff]↵
angela.martin@dunder.mifflin.com [staff]↵
bob.vance@vance.refrigeration.com [guest]↵
michael.scott@dunder.mifflin.com [manager]↵
pam.beesly@dunder.mifflin.com [staff]↵
create angela.martin@dunder.mifflin.com↵
The launch party↵
mid 2024 05 31 17↵
website launch scranton↵
The launch party is scheduled.↵
create michael.scott@dunder.mifflin.com↵
Disability awareness meeting↵
mid 2024 05 31 17↵
disability↵
Disability awareness meeting is scheduled.↵
```

```
create michael.scott@dunder.mifflin.com↵
Integration celebration↵
high 2024 04 28 14↵
merger stamford scranton↵
Integration celebration is scheduled.↵
invite pam.beesly@dunder.mifflin.com↵
angela.martin@dunder.mifflin.com The launch party↵
pam.beesly@dunder.mifflin.com was invited.↵
invite andy.bernard@dunder.mifflin.com↵
angela.martin@dunder.mifflin.com The launch party↵
andy.bernard@dunder.mifflin.com was invited.↵
response michael.scott@dunder.mifflin.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
accept↵
Account michael.scott@dunder.mifflin.com was replied accept to the invitation.↵
invite andy.bernard@dunder.mifflin.com↵
angela.martin@dunder.mifflin.com The launch party↵
andy.bernard@dunder.mifflin.com was invited.↵
response andy.bernard@dunder.mifflin.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
accept↵
andy.bernard@dunder.mifflin.com has replied accept to the invitation.↵
The launch party promoted angela.martin@dunder.mifflin.com was rejected.↵
response bob.vance@vance.refrigeration.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
reject↵
bob.vance@vance.refrigeration.com has replied reject to the invitation.↵
```

The following errors may occur:

1. One or both of the accounts do not exist (Account <non-existing account name> does not exist.), where the account name reported is the first found not to be registered.
2. The **response** to the event is unknown (Unknown event response.).
3. The event with **event name** does not exist in the **account name** (<event name> does not exist in account <account name>.).
4. The **account invitee** is not on the invitation list for the event (Account <account invitee> is not on the invitation list.).
5. The **account invitee** has already responded to the invitation (Account <account invitee> has already responded.).

```
... assume the above commands are executed ...
response grogu@mandalorian.com↵
david.wallace@dunder.mifflin.com Plenary meeting↵
reject↵
Account grogu@mandalorian.com does not exist.↵
```

```
response michael.scott@dunder.mifflin.com
david.wallace@dunder.mifflin.com Plenary meeting
maybe
Unknown event response.
response david.wallace@dunder.mifflin.com
angela.martin@dunder.mifflin.com Integration celebration
accept
Integration celebration does not exist in account angela.martin@dunder.mifflin.com.
register meredith.palmer@dunder.mifflin.com staff
meredith.palmer@dunder.mifflin.com was registered.
response meredith.palmer@dunder.mifflin.com
michael.scott@dunder.mifflin.com Integration celebration
reject
Account meredith.palmer@dunder.mifflin.com is not on the invitation list.
response pam.beesly@dunder.mifflin.com
michael.scott@dunder.mifflin.com Integration celebration
accept
Account pam.beesly@dunder.mifflin.com has already responded.
```

3.9 Command **event**

Shows detailed information of an event. The command receives as arguments the **account name** of the promoter and the **event name**. If successful, the command displays header line (<event name> occurs on <event date and time>:), followed by a list of invitees, one by line, in order of invitation, annotated with information about the invitation status (`accepted`, `rejected`, `no_answer`). To format the output of the date use the class `DateTimeFormatter` with the pattern `"dd-MM-yyyy HH'h'"`.

```
event michael.scott@dunder.mifflin.com Integration celebration
Integration celebration occurs on 28-04-2024 14h:
michael.scott@dunder.mifflin.com [accepted]
pam.beesly@dunder.mifflin.com [accepted]
angela.martin@dunder.mifflin.com [accepted]
bob.vance@vance.refrigeration.com [rejected]
david.wallace@dunder.mifflin.com [no_answer]
```

The following errors may occur:

1. The account with **account name** does not exist (Account <account name> does not exist.).
2. The event with **event name** does not exist in the **account name** (<event name> does not exist in account <account name>.).

```
... assume the above commands are executed ...
event grogu@mandalorian.com Plenary meeting
Account grogu@mandalorian.com does not exist.
event angela.martin@dunder.mifflin.com Integration celebration
Integration celebration does not exist in account angela.martin@dunder.mifflin.com.
```

3.10 Command `topics`

Shows all events that cover a list of topics. This command receives as argument a **`list of topics`** and always succeeds. If the application does not have any event that covers any the topics on the **`list of topics`**, the output is (No events on those topics.). Otherwise, it will present a header line (Events on topics <list of topics>:) followed by the events that cover at least one the topics, one by line, with the following format (<other event name> promoted by <promoter account> on <list of topics>), where the <list of topics> is the event topics presented by insertion order. The events should be ordered by the number of common topics and draws should be ordered by alphabetical order, first by the event description and then by the promoter email. The following scenario illustrates its usage.

```
topics slough uk↵
No events on those topics.↵
topics stamford merger↵
Events on topics stamford merge:↵
A working group event promoted by david.wallace@dunder.mifflin.com on stamford merge↵
Integration celebration promoted by angela.martin@dunder.mifflin.com on merger future stamford↵
Integration celebration promoted by michael.scott@dunder.mifflin.com on scranton merger stamford↵
Plenary meeting promoted by david.wallace@dunder.mifflin.com on merger↵
```

4 Project Development Methodology

This section provides some recommendations on a suitable work methodology for a software system development project. It is very important to be methodical in any activity of any duration, particularly in a project of this type. It entails a sequence of activities that must be carried out in a disciplined manner (i.e., a development process), to obtain a quality final "product", reducing the occurrence of defects (bugs) and facilitating their detection when they occur, easing the readability of the program, and... obtaining a good grade.

You can start by developing the main user interface of your program, clearly identifying which commands your application should support, their inputs and outputs, and error conditions. Then, you need to identify the entities required for implementing this application. Carefully specify the **interfaces** and **classes** that you will need. You should document their conception and development using a class diagram, as well as documenting your code adequately, with JavaDoc.

It is a good idea to build a skeleton of your `Main` class, to handle data input and output, supporting the interaction with your program. In an early stage, your program will not really do much. Remember the **stable version rule**: do not try to do everything at the same time. Build your program incrementally, and test the small increments as you build the new functionalities in your new system. If necessary, create small testing programs to test your classes and interfaces.

Have a careful look at the test files, when they become available. You should start with a really bare bones system with the `help` and `exit` commands, which are good enough for checking whether your commands interpreter is working well, to begin with. Then, implement the account registration. Once user account are correctly registered and listed, move on to scheduling and listing events. And so on. Step by step, you will incrementally add functionalities to your program and test them. **Do not try to make all functionalities at the same time. It is a really bad idea.**

Last, but not the least, **do not underestimate the effort for this project**.

5 Project submission

The program must be submitted to the Mooshak contest POO2324-TP2, by 5:00 pm, May 31, 2024 (Lisbon time).

Each group of 2 students must register in the contest. See the registration rules for Mooshak on the course page. Only programs submitted to Mooshak by users following these rules will be accepted to evaluation. For the project to be evaluated, students must have complied with the DI code of ethics (see the course page), and have, at the end of the deadline, a submission to the Mooshak contest. You can submit your code more than once. Only the last submission will be evaluated.

The project evaluation has 3 components:

1. Correctness of the produced results evaluation: up to 8 points.

   (a) Contest submission having the maximum score (100 points), where the concepts of *interface*, *inheritance*, *exceptions*, and *enumerations* defined in lectures are used, will have 8 points in this component.

   (b) A submission with errors in some of the features will have a classification corresponding to correctly implemented features.

   (c) If the submitted program does not use the concepts of *interface*, *inheritance*, *exceptions*, and *enumerations*, the score is reduced up to half of that obtained in (a) or (b).

2. Code quality evaluation: up to 12 values.

3. Written discussion (see rules on the course page or Clip).