Home exam (hjemmeeksamen)

PGR112 Object-oriented Programming

August 2023

Home exam, individual

Notice:

- The main topics in this exam are **Java**, **JDBC**, and the use of **Object-oriented programming** as shown in examples through the course.
- You are encouraged to use code from the course related git repository (https://github.com/bogdanmarculescu/pgr112oop) or from any of the lecture slides, but you **must add code that addresses the specific task requirements**. If you do use existing code, you must clearly show (in comments) what the source of that code was.
- You should put all the files that the delivery contains in one folder and **zip** before uploading to WISEFlow.
- The solution should be able to run properly in IntelliJ (i.e. not have to run with any other special technology/IDE)
- The solution submitted **must** compile and run. This also means the submission **must** include all relevant source code files.
- If you are unsure about something regarding the exam, re-read the instructions and try to make a best assumption. Clearly express what assumptions you made and why in the report (see below).

The delivery will contain three things:

1. The project folder with Java, folders, txt or json files if any, etc. The delivery **must** contain the source code files (.java) together with all necessary files to successfully compile and run the project.
2. The database schema files you have created (see SQL files in repository for examples).
3. A small report (pdf or word document) that shortly describes the major OOP concepts you have implemented and how you have used them through your design.

Case – Voting for årets medstudent

In the context of a university graduation ceremony, your task is to implement a system that handles voting for the "Student of the year" award (Årets medstudent).

Students in each study program can vote for one of their fellow students to be "årets medstudent". In addition to their vote, each student may add a small comment about the person they voted for, and these comments may be read aloud at the ceremony.

Your task is to design and develop a small system that allows for this election to take place.

A user can:

- See the current **Nominees**
- Vote for one of the **Nominees**, adding their vote to the total number of votes for that **Nominee**.
    - Only someone who is a **Student** may vote.
    - A **Student** may only vote **once.**
    - A **Student** may add a small comment about the **Nominee** they voted for.
- If the person that they wish to vote for is not on the list, they can propose a new **Nominee.**
    - A new **Nominee** must not have been Nominated before (that is to say, a **Student** may only be nominated for this once).
    - A **Student** may NOT nominate themselves.
- See the current number of votes and the comments about each **Nominee.** But a student may not see who voted or proposed each comment.
- See who is currently winning – the **Nominee** with the highest number of votes, and the comments that were attached (again, without seeing who proposed each comment).
- Only a **Student** can vote or nominate someone. (Hint: this could be checked, for example, by getting a user to input a unique student id number).

Additional requirements. If the other requirements are complete, you can try to add some more relevant functionality. For example, a user can:

- Only vote for **Nominees** that are in the same study program.
- Only propose a new **Nominee** from the same study program as the user themselves.

You are free to add functionality as you find interesting and relevant, and to the extent to which time allows for it. I would advise that additional functionality is only added after you are confident that the required functionality is complete. I also advise that additional functionality be described in the Readme file, so that the evaluator can be aware of it.

In addition (for testing and evaluation):
A user (censor):

- Can get a list of all students (so appropriate examples can be selected for testing and evaluation).

Databases and tables

- You should create (and populate with relevant data) a database for students (for example called **studentDB**), containing all students and the study programs they are registered for. **Hint**: to allow you to test that only a student from a relevant study program may nominate someone for the award, it may be useful to have several students from at least 2 study programs as initial test data.
    - This means that everyone who is a **Student** is already registered in that database.
    - Only a **Student** may vote, propose a new **Nominee**, or be a **Nominee.**
    - Your program is not expected to add or delete information in the **studentDB** database (your are just keeping track of votes). Hint: this means that you

    should probably fill in test data in this database through an appropriate sql script

- You should create a separate database to keep track of the votes and comments. (This could be named something like **voteDB, electionDB,** or anything else you find appropriate).
    - This database will keep track of **Nominees**, and will be updated when
        - A new **Nominee** is proposed
        - A new vote is registered
        - A new comment is added
- You are free to add any additional tables, databases, scripts or items you may need.

Object-Oriented Program

- You should create an object-oriented program that handles the interaction with the user (see below **Interactive Component**) and with the database (see above **Databases and tables**)
- It is expected that you focus on using object-oriented principles when writing the code that solves the case described above. There is no single right answer, and you have the freedom to solve the problem as you see fit, but object-oriented programming is a fundamental requirement of this course.
- The program will have different types of objects that house the appropriate data and functionality.

Interactive Component

- Interaction with the users will be handled through Java.util.Scanner, as presented in class.
- The interface should present options that allow a user to fulfill the requirements presented above. This includes (but is not limited to):
    - Allow a user to identify themselves (for example using the student id).
    - Display a list of current **Nominees**
    - Allow the user to propose a new **Nominee**
    - Allow the user to vote for one of the **Nominees**
    - Allow the user to add a comment to the **Nominee** that they voted for (this could be done at the same time as the vote, or separately – pick the option which you prefer).
    - See how many votes each **Nominee** has
    - See the current top **Nominee** (that is the **Nominee** with the highest number of votes) and the comments that **Nominee** received.

NOTE:
A **Nominee** – is a **Student** that has been nominated (proposed) by one of their colleagues.
Hints:

- A **Nominee** class object can be used to keep track of the number of votes and of the comments that the individual received.
- A **Nominee** can be a type of **Student** – if you want to use Inheritance, or can contain a reference to a **Student** – if you want to use Aggregation. In both cases, a **Nominee** has additional information as compared to the **Student** class.

- The **Nominee** class may contain functionality regarding keeping track of votes and comments, and returning such information when needed. (For example, each **Nominee** keeps track of their own votes and comments).

Examples:
In the following example, program outputs are shown in blue, and the user inputs are shown in red and marked with a "->" symbol

```
Welcome! Here are your options:
    1.  Sign in
    2.  See the current top Nominee
    3.  Exit

-> 1

Student id:
-> 42

Welcome, Bogdan! You have not voted yet. Here are your options:
    1.  See all Nominees
    2.  Propose new Nominee
    3.  Vote and add a comment for a Nominee
    4.  Return to main menu.

-> 1

Current Nominees are:
    1.  Per
    2.  Helene
    3.  Lars

Welcome, Bogdan! You have not voted yet. Here are your options:
    1.  See all Nominees
    2.  Propose new Nominee
    3.  Vote and add a comment for a Nominee
    4.  Return to main menu.

-> 2

Proposing a new Nominee. Insert student id:
-> 1138

You have Nominated Anne.

Welcome, Bogdan! You have not voted yet. Here are your options:
    1.  See all Nominees
    2.  Propose new Nominee
    3.  Vote and add a comment for a Nominee
    4.  Return to main menu.

-> 3

Current Nominees are:
    1.  Per
    2.  Helene
    3.  Lars
    4.  Anne
Who do you wish to vote for? (select number):

-> 4

Registered your vote for Anne. Would you like to add a comment (y/n)
-> Y

Add your comment:
-> Motivated and very knowledgeable.

Welcome, Bogdan! You have voted. Here are your options:
    1.  See all Nominees
    2.  Propose new Nominee
    3.  Change your comment for
    4.  See Nominee with the highest number of votes
    5.  See the votes for all Nominees.
    6.  Return to main menu.

-> 4
Nominee with the highest number of votes at the moment is:
---
Anne – 1
"Motivated and very knowledgeable"
---
```

NOTE: This example is just for reference. Feel free to adjust how the program shows and collects information, how much detail you want to include, and how complex you want the interaction to be.

So long as your program fulfills the requirements described above, you are free to adjust the interface as you see fit.

Other details, hints, advice

- Start small, with the most fundamental functionality you think you will need.
- Add more functionality as needed and relevant, once existing functionality **works**.

- Look for opportunities to use **Object-Oriented concepts**, where relevant.
- Start with the fundamental requirements (using basic object-oriented concepts , reading and writing information to/from database, interacting with the user via console) before moving to the more advanced ones.
- Clarify (for example, in a readme.MD file) any assumptions that you make, how to run your project, and any other information you want to provide to users/evaluators
- Your project must **compile** and **run.**
- You should use the technologies in the course. (It is not reasonable to expect the evaluator to install and run a variety of different databases and libraries to accommodate individual submissions).

Assessment criteria, for students and assessors

The following will be the main points to assessed:

- Java, Object-oriented programming, and JDBC is what is assessed in this exam.
- The basic OOP concepts such as Inheritance, Polymorphism, Abstraction and Encapsulation.
- Data types such as ArrayList, HashMap, etc.
- Use of Java:
  - Basic use: correct usage of basic datatypes, correct use of objects and dynamic code, correct use of methods, return values and parameters.
  - Handling of exceptions
  - Input validation
  - Advanced use: Iterators, correct use of Aggregation, other relevant techniques
- Amount of functional code and complexity – the code should be of suitable complexity to fulfil all the requirements in the case, but not more complex than is needed and readable.
- Folder structure, tidy code, following naming conventions, clarity and readability of the code.

Guideline for how much each part counts

The percentage numbers below present approximate numbers on how much each part will count

- The degree to which the submission fulfils the case requirements:	ca. 20%
- Appropriate use of OOP concepts:	ca. 20%
- Use of JDBC:	ca. 20%
- Use of java (data types, basic use, advanced techniques, exception handler etc):	ca. 15%
- Report (with focus discussing OOP concepts from own experience and with examples from own code):	ca. 10%
- Code structure, tidy code, good names on variables and functions, code readability and understandability etc.:	ca. 10%
- *Amount of code and complexity:	ca. 5%

*The percentages are only guidelines, as all points affect each other. The assessment will require a holistic view of techniques applied in the exam delivery and an overall evaluation of the degree to which the submission meets the requirements and shows appropriate understanding and use of Object-Oriented Programming.

--End of exam text--