PGR112, HOME EXAM, 24H, INDIVIDUAL

Notice:

- The main topics in this exam are:
  - Java
  - JDBC
  - Object-oriented programming theory

- All code relevant to the solution must be developed by you. Except for in very special and concrete cases, you **cannot** copy and paste code that is not yours. You can use existing code available in the course Git-repository as a reference. If you do use existing code, you must clearly show (using comments) what and where the source of that code is.

- You should put all the files that are a part of your delivery in one folder and create a .zip-archive which is to be uploaded to WiseFlow as your exam delivery.

- The solution should be able to run properly within IntelliJ (for example, not having to use any other special IDE or technology to run your code).

Make sure to read all pages of this exam text, especially the pages regarding **workload** and **details** relevant to this exam, as these are intended to help you understand what the exam is asking for.

If you are unsure about anything regarding the exam, re-read the instructions and try to make a best assumption. Clearly express what assumptions you made and state why in the report (see below for more information about the report).

The delivery will contain three parts:

1. The project folder that contains all of your .java-files, usually in a \`src\`-folder.
2. The database (.sql) files needed to setup the database, tables and any initial data
3. A small report (.pdf or word document) (more details on the bottom of page 2)

Pizzeria: Menu suggestions from customers

You are given the task to implement an interactive program for a pizzeria that is looking to collect suggestions from customers as to what different kinds of new pizza will be on the new menu at this pizzeria. This interactive terminal is shut down when the pizzeria closes and is started up again at the start of the next day.

In this system, all pizzas consists of three components which make up a pizza:

1) The dough
2) The sauce
3) A list of different toppings

A suggestion for a new pizza contains the following:

1) A name for the pizza
2) The three components of that pizza
3) An email belonging to the person that made the suggestion

The menu should initially have at least three options:

1) Submit new pizza *(An example is given on page 7)*
2) Display overview of doughs, sauces and toppings that exist
3) Display existing suggestions

This interactive program, when it starts, it should load all suggestions, pizzas and their components (doughs, sauces and toppings) made so far from the database into data structures which is to be used by the program.

You have to initialize your database with at least:

- 3 doughs
- 2 sauces
- 5 toppings

  You can use a real pizzeria menu for inspiration here if needed.

In addition, you shall make a report (200-400 words) in a .pdf- or word-document which answers the following:

1) What is Object-oriented programming (OOP)?
   Which OOP concepts did you use in your solution?
   How did you use these concepts to develop your solution?
2) What is JDBC? When did you use it? How did you use it?
3) Concrete examples of OOP-concepts used in your code delivery.

Workload:

- Database and schema
  - You must create a database named **pizzeria_suggestions**
  - You must create the following tables and columns:
    - **suggestion**
      - *id; pizza_id;* pizza*_name; email; count*
    - **pizza**
      - *id; dough_id; sauce_id; toppings*
    - **pizza_dough**
      - *id; name; description*
    - **pizza_sauce**
      - *id; name; description*
    - **pizza_topping**
      - *id; name; description*
    - **duplicate**
      - *id; suggestion_id;* pizza*_name, email*
  - These tables are updated based on input given in the terminal.
  - The data in these tables are loaded into the program when it starts.
  - The program contains all class instances created, for example, when a new **pizza**-record is inserted into the pizza-table, you should also initialize a new instance of the **Pizza**-class and store this instance in a data structure within the program. This goes for all of the tables within the database.
- Interactive Java-progam
  - You must use `java.util.Scanner` to build the interactive console interface.
  - You must be able to do the following using a menu:
    - Submit a suggestion
    - View all existing suggestions
    - List all types of dough, sauce and different toppings
    - If not at the main menu (first menu options shown), the ability to go back to the start of the program

Workload (*continued*):

- Static methods
  - You are to implement the following functionality available as static methods in the class representing suggestions within the program:
    - `getAllSuggestionsWithDough(int dough_id): ArrayList<Integer>`
      - Returns a list of suggestions with specified dough
    - `getAllSuggestionsWithSauce(int sauce_id): ArrayList<Integer>`
      - Returns a list of suggestions with specified sauce
    - `getAllSuggestionsWithTopping(int topping_id): ArrayList<Integer>`
      - Returns a list of suggestions where the pizza has the given topping as one of the toppings that makes up that pizza
    - `getAllNamesForSuggestion(int suggestion_id): ArrayList<String>`
      - Returns all pizza names connected to one suggestion using the suggestion id given to this method

        Hint: The **duplicate**-table contains alternative pizza names for suggestions that already exist
    - `getMostSuggestedPizza(): ArrayList<Integer>`
      - Returns all pizza ids which represents the pizzas that has received the most duplicate suggestions

        Hint: This is what the *count*-value in **suggestion**-table is for
    - `getAllUniqueSuggestions(): ArrayList<Integer>`
      - Returns a list of suggestions (id) that has no duplicates

Details

- You can create a PizzaComponent class as an abstract class or interface, which PizzaDough, PizzaSauce and PizzaTopping can extend from.

- The following classes are expected as a minimum:
  - PizzeriaMenuSuggestions
  - Suggestion
  - Duplicate
  - Pizza
  - PizzaDough
  - PizzaSauce
  - PizzaTopping

- The PizzeriaMenuSuggestions-class acts as the program which manages the interaction between the customer and the suggestion-taking-system: A system which loads data from a database and displays the console interface in a terminal.

- If a new suggestion contains a pizza name that already exists within the suggestion-table or duplicate-table, the customer is to be prompted for a new pizza name instead.

- The *toppings*-column in the pizza-table is a text value representing a comma-separated list, for example: "1,2,4". Each of the numbers represents an integer value which represents each topping id that makes up a pizza.
  - Before inserting this *toppings*-value into the pizza-table, you have to create this comma-seperated list first.

    Hint: it can be beneficial to sort the number values before creating the text representation of this list of ids to be inserted.

  - Before creating an instance of PizzaTopping, you have to split up this *toppings*-value received from the database to know which id-values are within this comma-separated list stored as a text value.

- Make sure you try to include the different OOP-concepts, such as:
  - Inheritance, abstraction, encapsulation, and composition

- Communication with the database:
    - When the program starts, load all current information from the database into the program

    - It can be beneficial to also store the id (primary key) for the diffferent tables from the database in the class instances stored within the program.

    - When a suggestion for a new pizza is made, you have to check if a record in the pizza-table contains the exact same pizza components specified in the suggestion, and:
        - If a pizza with the same components already exists:
            - Find the existing ***suggestion***-record in the **suggestion**-table and update the *count*-value by adding one (1) to existing value.

            - If the suggested pizza name is different from the pizza name in the existing suggestion, create a new record in the **duplicate**-table containing the id of the existing suggestion.

        - else, as a suggestion for a new pizza that does not already exist:
            - Create a new record in the pizza-table with the components from the suggestion.

            - Create a new record in the suggestion-table containing the id of the ***pizza***-record created above, the name of the pizza, the email of the suggestion creator and a count-value initially set to one (1).

              Hint: Instead of looking in the database for the pizza newly created for the pizza id to use here, when creating this new record in the suggestion-table, use the largest pizza id plus one (1) from instances stored inside the program.

As an **example**, the terminal menu could function as this when submitting a new suggestion:

1) Main menu:
    a. Submit a suggestion for a new pizza
    b. View suggestions made by other customers
    c. View list of doughs, sauces and toppings

   Customer enters “A” into the terminal

2) Select a type of dough:
    a. New York-style
    b. Neapolitan-style
    c. Sicilian-style

   Customer enters “B” into the Terminal

3) Select a type of sauce:
    a. Tomato
    b. White

   Customer enters “A” into the terminal

4) Select toppings:
    a. Pepperoni
    b. Ham
    c. Cheese
    d. Onion
    e. Go to next step …

   Customer enters “B” into the terminal

5) Select toppings:
    a. Pepperoni
    b. Cheese
    c. Onion
    d. Go to next step …

   Customer enters “B” into the terminal

6) Select toppings:
    a. Pepperoni
    b. Onion
    c. Go to next step …

   Customer enters “C” into the terminal

```
7)  Pizza combination:
    - Dough: New York-style
    - Sauce: Tomato
    - Toppings: Ham, Cheese
    Create a name for your pizza suggestion:
    Customer enters "Ham & Cheese" into the terminal

8)  Suggestion by you:
    - Name: Ham & Cheese
    -- Dough: New York-style
    -- Sauce: Tomato
    -- Toppings: Ham, Cheese
    Enter your e-mail to submit your suggestion:
    Customer enters "customer@pizza.place" into the terminal

9)  Thank you for your submission!
    We will contact you if we end up with this pizza on the menu!

    Press any key to go back to the main menu ...
```

Note that the above example is just that, just an example. You are free to shape how the program will behave and how information is displayed within the terminal window. You can add as much details and complexity as you can, but be sure to first solve what the task asks you for in terms of the workload specified on page 3 & 4 of this exam text.

Assessment criteria, for students and assessors

<u>The following will be the main points to be assessed</u>:

- Java, JDBC and Obect-oriented programming is what is assessed in this exam.
- The basic OOP concepts such as:
    - **Inheritance**, **Polymorphism**, **Abstraction**, **Encapsulation**
- Use of data structures such as ArrayList, HashMap, etc.
- Use of advanced Java-techniques such as: Iterators, Optional and lambda-expressions.
- Exception-handler to handle exceptions which may occur, combined with input validation to validate user input
- Amount of (good) code and complexity are of importance
- Folder structure, tidy code, naming convention for classes, variables and methods

<u>Guideline for how much each part counts</u>:

The percentage numbers shown below presen approcimate numbers on how much each part til count:

- *User-interface logic*:	ca. 20%
- *Use of OOP-concepts*:	ca. 20%
- *Use of JDBC*:	ca. 20%
- *Java techniques*:	ca. 15% (data type, exception handler, etc.)
- *Report*:	ca. 10%
- *Code in general:*	ca. 10% *(stucture, tidy code, naming convention, etc.)*
- *Amount of code and complexity*:	ca. 5%

Be aware that all parts and points affect each other. The assessement will require a holistic view of techniques applied in the exam delivery.

--- END OF EXAM TEXT ---