Advanced Object Oriented Programming-2 (PROG36859)

Assignment 3

Due Date: See Slate

INSTRUCTIONS

- This assignment must be completed in a **group of at most 2 students** without any outside collaboration (Students should form their own groups). All work must be your own group. Copying or reproducing the work done by others (in part or in full) or letting others to copy or reproduce your own, and/or unauthorized collaboration will be treated as academic dishonesty under the College's Academic Dishonesty Policy.
- **IMPORTANT**: You must submit screenshot(s) demonstrating your work as instructed in the submission guideline. All screenshots MUST show your name and student Id (you can place a small text window containing your name and id on top of your screen, not covering any content.) All screenshots must be readable in 100% zoom size. Your submission may not be marked (or receive significant grade cut) if one or more required screenshot is missing or unreadable.
- Your application(s) must compile and run upon download to receive any mark. Your supplied outputs (e.g., screenshots) may not be marked, if the corresponding program fails to compile and run.
- To submit the assignments, please follow the Submission Guideline provided at the end of this assignment.
- You must submit the assignment by the due date. Late submissions policy is specified in the Course Plan document available on Slate.
- Total mark = 100 (weight = 10% of the final grade).

Feed Zoo Animals

Table 1 Animal Food Chart

Animal	Required amount of food (kg) in stock
Hippo	15
Rhino	12
Lion	9
Zebra	5
Monkey	3

Simulate a Zoo Animal Feeding System in Java. Assume the system maintains a stock of food and takes care of feeding for five different animals, which are hippo, rhino, lion, zebra, and monkey. Each of these animals has different amount of food requirements. An example of food requirements per animal is shown in Table 1). For example, at least 15 kg food is required in the food stock to feed a hippo, whenever it gets hungry.

Feeding Task [*Requirement 1*]:

Animals become hungry at a random order, but one animal at a time (i.e., if an animal gets hungry, then no other animal can be hungry until the hungry animal is fed its required amount of food.) If the required amount of food for the hungry animal is available in the food stock, the

hungry animal is fed right away, and the food stock is updated with this food consumption. If the required amount of food to feed the hungry animal is not available in the stock, the Feeding Task system must wait for the Depositing Task (described below) system that adds food to the food stock.

Depositing Task [*Requirement 2*]:

Food is deposited to the food stock at a random amount between 1 kg and 20 kg, at a time.

Concurrency [*Requirement 3*]:

Both tasks (Feeding Task and Depositing Task) run concurrently, but in a synchronized manner. When the Feeding Task consumes food from the stock, the food stock is locked to the Depositing Task. Similarly, during the time the Depositing Task adds food to the stock, the food stock is locked to the Feeding Task. However, after each deposit, the Depositing Task should send signal to (all) waiting Feeding Task for possible food consumption.

Processing Feeding Data [*Requirement 4*]:

Your simulation will keep track of number of animals that have been fed. It will terminate both tasks of Feeding and Depositing, after feeding *n* animals (*n* is the total number of individual animal to be fed by the system, and may be supplied by the user, or pre-defined). It will also keep track of the total amount of food fed to each specific animal.

Storing Data in Database [*Requirement 5*]:

Create a database named ZooDB_<Your Student Id> on you MySQL database through MySQL terminal/workbench. Once the feeding process is done, your simulation should connect to ZooDB_<Your Student Id> database and create a table called FeedingData (animalName, feedingCount, hungryCount), and store the feeding information for all the animals into the table using prepared statement. The animalName field should store the name of the animal and feedingCount field should store how many times that animal was fed, and hungryCount field should store the total number of times the animal got hungry.

Processing Data from Database [*Requirement 6*]:

Your simulation will also display following information through collecting required information from the database:

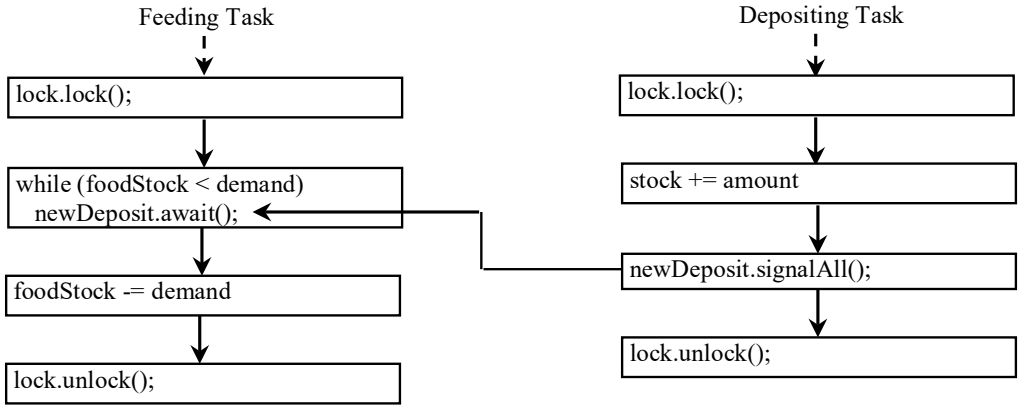
(i) Full information from FeedingData table along with column headings (Hint: use ResultSetMetadata to collect column information),
(ii) The animal that consumed the highest amount of food. (Hint: calculate it by using the value in feedingCount and required unit food for an animal). If there is a tie, display all tied animals,
(iii) The animal that got hungry most of the time (i.e., hungryCount). If there is a tie, display all tied animals, and
(iv) Total amount of food consumed by all *n* animals.

[**IMPORTANT**: You **must** retrieve above information from the database table and retrieve metadata using available classes for getting metadata.]

Development:

Develop your Java program to demonstrate thread cooperation through a thread pool for the Feeding Task and Depositing Task. Suppose that you create and launch two threads, one of them adds food to the food stock (i.e., Depositing Task), and the other consumes food from the same food stock (i.e., Feeding Task). The second thread must wait, if the amount of food to be consumed is more than the current stock. Whenever new food is added to the stock by the first thread, the first thread notifies the second thread to resume. If the amount is still not enough for a consumption, the second thread must continue to wait for more food in the stock. Assume that the initial food stock is 0, and that the amount of food to be deposited and the selection of animal to be fed (i.e., the amount of food to be consumed) are randomly generated.

To synchronize the operations, use a lock with a condition: *newDeposit* (i.e., new food added to the stock). If the stock is less than the amount to be consumed, the Feeding Task will wait for the *newDeposit* condition. When the deposit task adds food to the stock, the task signals the waiting Feeding Task to try again. The interactions between the two tasks are shown in the following figure.



Handling the database operations and query processing can be done through a separate standalone thread (not within the thread pool), which should be started after finishing the Feeding Task (i.e., after shutting down the thread pool).

Output:

A sample output for a typical run for *n* = 10 is shown below. The left most column displays the (random) amount of food added by each deposit task, the middle column displays animal feeding/waiting for food information from the feeding task, and the last column shows the status of food stock after each operation of depositing task and/or feeding task. Because this simulation relies on threads, thread synchronization, depositing a random amount of food to the food stock, selecting a random animal from the animal list, and the output from different run may vary (significantly). However, in every scenario, the thread pool should terminate after feeding *n* animals. For example, in the following output, the thread pool terminates after feeding 2 rhinos, 2 hippos, 1 zebra, 3 monkeys and 2 lions (total 10 animals) selected in random order.

Deposit Food	Feed Animals	Stock (kg)
Add 3kg		3
	Zebra got hungry, Wait for food..	
Add 10kg		13
	Rhino got hungry, Feed Rhino 12kg	1
	Monkey got hungry, Wait for food..	
Add 4kg		5
	Rhino got hungry, Wait for food..	
Add 1kg		6
	Hippo got hungry, Wait for food..	
Add 15kg		21
	Hippo got hungry, Feed Hippo 15kg	6
	Rhino got hungry, Wait for food..	
Add 12kg		18
	Hippo got hungry, Feed Hippo 15kg	3
	Monkey got hungry, Feed Monkey 3kg	0
	Lion got hungry, Wait for food..	
Add 3kg		3
	Monkey got hungry, Feed Monkey 3kg	0
	Lion got hungry, Wait for food..	
Add 3kg		3
	Zebra got hungry, Wait for food..	
Add 10kg		13
	Rhino got hungry, Feed Rhino 12kg	1
	Rhino got hungry, Wait for food..	
Add 8kg		9
	Lion got hungry, Feed Lion 9kg	0
	Zebra got hungry, Wait for food..	
Add 9kg		9
	Hippo got hungry, Wait for food..	
Add 5kg		14
	Zebra got hungry, Feed Zebra 5kg	9
	Hippo got hungry, Wait for food..	
Add 3kg		12
	Lion got hungry, Feed Lion 5kg	7

```
                Rhino got hungry, Wait for food..
Add 3kg                                                          10
                Monkey got hungry, Feed Monkey 3kg               7
Finished FeedingTask thread....
Finished DepositingTask thread...


Query result from DBOperation thread...
     1. All rows from FeedingData table:
          animalName feedingCount hungryCount
          Monkey     3            4
          Hippo      2            5
          Rhino      2            6
          Lion       2            4
          Zebra      1            4

     2. Highest amount of food consumed by: Hippo
     3. The hungriest animal: Rhino
     4. Total food consumed by all 10 animals: 86kg
```

Important: [*Requirement 7*]

a. You must display following information in your main class file as commented:
   - Course: PROG36859
   - Assignment No.: 3
   - Member 1: <Your name>, <Student Id>
   - Member 2: <Your name>, <Student Id>
   - Instructor's name: Syed Tanbeer

b. Your program must generate all outputs programmatically and work for a different amount of food requirement per animal (do not hard code data).

c. You must demonstrate the use of thread synchronization, lock and condition, thread, thread pool, prepared statement, metadata.

d. You must use at least one class (i.e., Animal class with required properties to process food requirement) in your program. Use more classes, if required.

e. You must follow the development process mentioned under Development section and naming convention for naming various components (e.g., database name).

f. You must demonstrate Java code writing standard, and modularization in your code.

Submission Guideline:

- Create a document file named <Member1FirstName-Member2FirstName-A3>. The cover page of this document must include following information:
  - Assignment #: 3
  - Course: PROG36859
  - Member 1: Full Name (Student Id)
  - Member 2: Full Name (Student Id)
  - Instructor's name: Syed Tanbeer

- Run your application and take screenshot showing outputs (as shown above) from **2** sample runs of your program with two different values for *n* (e.g., 10 and 15).
- Open your database table on your MySQL terminal/workbench showing its contents and take a screenshot that shows the table contents, table name, database name, and the commands issued.
- All screenshots must satisfy the requirement as indicated in the INSTRUCTION of this assignment. Paste the screenshots in the document file in sequence with reference number. Screenshot details and mark breakdown are presented in the following Table 1 below.
- Submit following files to the ''Submit Assignment 3" Dropbox available at Slate.
    - Whole project as a single zip file (e.g., .zip). [Note: Remove the MySQL database connection parameters (i.e., url, your username, and your password) from your code. To run your program, only this part of your code will be changed to connect to the MySQL system hosted in the machine to be used for evaluation purpose.]
    - The document file (Word or pdf) containing screenshots showing outputs from a sample run of your simulation (as shown above) and database contents.

[Note: Your program will not be marked, if any required file in your project or the additional document file is missing, corrupted, or it fails to open. Please double check whether your zip conversion is successful, by unzipping it and opening all files at least once, before submitting.]

Table 1: Marks breakdown and screenshot descriptions

Requirement #	Marks	Screenshot
1*	20	Output
2*	20	Output
3*	25	Output
4	5	Output
5*	10	Database
6	20	Output
7*	-	-

* Your submission may not be marked (and get zero) if this requirement is missing/incorrectly done.

---: End of Assignment 3 :---