Lab Test 2 [25 marks]

General Instructions:

1. **You are not allowed to communicate in any way during the test. Disable the following on your laptop before the test begins: wifi and any other communication devices (e.g. 3G/4G modems). Any violation of this instruction will result in a zero score for your lab test.**

2. You will perform the lab test on your personal laptop.

3. You can refer to any files on your laptop.

Failure to do the following will attract a penalty up to 20% of your score for that question.

4. Make sure your code can generate exactly the same output as we show in the sample runs. You may get some marks deducted for missing spaces, missing punctuation marks, misspelling, etc. in the output.

5. Do not hardcode. We will use different test cases to test and grade your solutions.

6. Follow standard Java coding conventions (e.g. naming of getter and setter methods, choice of identifier names for classes, methods and variables) as well as indent your code correctly.

7. Tools that enable AI pair programming (e.g., GitHub co-pilot) or pair programming (Live Share) are not allowed.

8. Ensure that all your Java code can compile without compilation errors. They must compile with any test class(es) that are provided. You may wish to comment out the parts in your code, which cause compilation errors. But remember that commented code will NOT be graded. **Only code without compilation errors SHALL be graded.** You may need to comment out parts of the test class' code that you have not implemented in order to test your implemented solutions.

9. Download the source code and rename the root folder in the zip file to your **Email ID (e.g. `jiagu.wen.2023`)**

10. **Include your name as author** in the comments of all your submitted source files. For example, if your registered name is "Jia Gu Wen", include the following block of comments at the beginning of each source file (.java) you write.

```
    /*
* Name     : Jia Gu Wen
* Email ID: jiagu.wen.2023
*/
```

11. Do **NOT** change the signature (parameter and return type) of the methods and DO NOT add in unnecessary packages.

12. You can

1. add additional private static helper methods within the given .java files to solve the questions
2. use Java streams, lambda expressions in the code.

Question 1a (*) [4 marks] (Modified from Practice-IT)

Implement a static method called `stutter` (`Q1a.java`). This method takes in 2 parameters:

- `s` (type: `String`): This is a non-null string.
- `n` (type: `int`): This is either 0 or a positive value.

This method will return the string with its characters repeated n times.

Question 1b (*) [4 marks] (Modified from Practice-IT)

Implement a static method called `percent` (`Q1b.java`). This method takes 2 parameters:

- `numbers` (type: `long[]`): This is the list of numbers.
- `isOdd` (type: `boolean`): A boolean value.

This method will return the percentage of odd numbers if isOdd is true. Otherwise, returns the percentage of even numbers.

IMPORTANT: Keep **Q1a.java and Q1b.java**
in the folder <your email ID>/Q1
Only these files will be marked, do not create additional java files

Question 2

Given:

1. the class diagram in the appendix,
2. the API documentation in the api folder.

Assumptions :

1. **The equals, hashCode methods are NOT implemented in Product, Perishable and Shop classes.**
2. **A Product is identified by the name attribute.**
3. **A Shop is identified by the name attribute.**
4. **A Shop will NOT have two Products of the same name, i.e. products will not store two Product objects (with the same product name) having Shop with the same name.**

Implement the following static methods.

Question 2a (*) [3 marks]

Implement a static method called getCheapestShop (Q2a.java). It takes in 2 parameters:

a. products (type : List<Product>): This is a list of Product objects. Not null.
b. name (type : String) : The product's name. Not null.

This method returns the cheapest Shop that sells this product. If there is more than 1 shop with the cheapest price, return the one with the **SMALLEST INDEX** in products.

Question 2b (*) [3 marks]

Implement a static method called getProductsOfCategory(Q2b.java). It takes in 2 parameters:

a. products (type: List<Product>): A list of Product objects. Not null.
b. category (type: String): The category of products that we are interested in. Not null.

You are **REQUIRED** to complete the necessary codes in ProductComparator.java. It compares two Product objects by their name in ascending order.

It returns the list of products that belong to the specified category. The output is sorted by the product's name.

Question 2c (**)[3 marks]

Implement a static method called getExpiringPerishables (Q2c.java). It takes in 2 parameters:

a. products (type: List<Product>): This is the list of Product objects. Not null.
b. n (type: int): The number of days. This is a positive value.

This method returns all the products that are Perishables whose expiry date is tomorrow or later but will expire within n number of days (inclusive of the n-th day).

Note : You should not hardcode today’s date.

Question 2d (***) [4 marks]

Implement a static method called getTheCheapestShops (Q2d.java). It takes in 1 parameter:

a. products (type: List<Product>): This is the list of Product objects. Not null.

This method returns a map where

- the key is the product name.
- the value is the list of shop names that sell this product with the lowest price.

Note:

1. The key is sorted by the product name in ascending order.

2. The value is sorted by the shop name in ascending order.

Question 2e (***) [4 marks]

Implement a static method called `getShopsWithHighestCommonProducts` (`Q2e.java`). It takes in 1 parameter:

a. `products` (type: `List<Product>`): This is the list of Product objects. Not `null`.

This method returns all the shops with the highest number of common products as a map where

- the key is the shop name, and
- the value is the list of shop names with the highest number of common products with the key. The value is sorted by the shop name.

The map is sorted by the key.

For example,

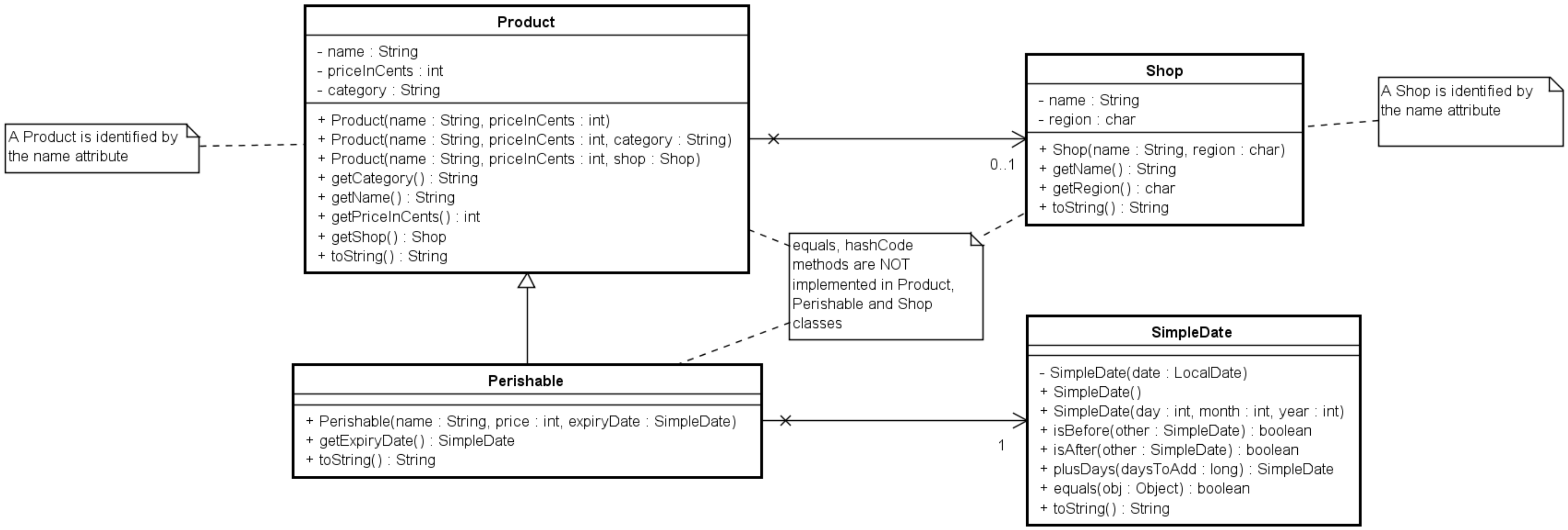
Shop	Products
S1	P2, P3, P4, P5, P6, P1
S2	P1, P2, P3
S3	P5, P4, P6
S4	P1, P2

This method will return `{S1=[S2, S3], S2=[S1], S3=[S1]}` since

- The highest number of common products is 3.
  - ○ S2 & S4 only have 2 common products.
  - ○ S1 & S4 only have 2 common products.
- S1 has 3 common products with S2 (P1, P2, P3) and S3 (P4, P5, P6).
- S2 has 3 common products with S1 (P1, P2, P3).
- S3 has 3 common products with S1 (P4, P5, P6).

IMPORTANT: Keep **Q2a.java, ProductComparator.java, Q2b.java, Q2c.java, Q2d.java and Q2e.java** in the folder <your email ID>/Q2
Only these files will be marked, do not create additional java files

APPENDIX


A Product is identified by the name attribute
Product
- name : String
- priceInCents : int
- category : String
+ Product(name : String, priceInCents : int)
+ Product(name : String, priceInCents : int, category : String)
+ Product(name : String, priceInCents : int, shop : Shop)
+ getCategory() : String
+ getName() : String
+ getPriceInCents() : int
+ getShop() : Shop
+ toString() : String
0..1
Shop
- name : String
- region : char
+ Shop(name : String, region : char)
+ getName() : String
+ getRegion() : char
+ toString() : String
A Shop is identified by the name attribute
equals, hashCode methods are NOT implemented in Product, Perishable and Shop classes
Perishable
+ Perishable(name : String, price : int, expiryDate : SimpleDate)
+ getExpiryDate() : SimpleDate
+ toString() : String
1
SimpleDate
- SimpleDate(date : LocalDate)
+ SimpleDate()
+ SimpleDate(day : int, month : int, year : int)
+ isBefore(other : SimpleDate) : boolean
+ isAfter(other : SimpleDate) : boolean
+ plusDays(daysToAdd : long) : SimpleDate
+ equals(obj : Object) : boolean
+ toString() : String