CS116 PROJECT SPRING 2021

By: Nabilah Siddiqui, Lydia Bellil, Gauri Kumari, and Muhammad Bilal Aijaz

2- or 3-person teams, one submission in Blackboard per team
Design due in Blackboard Friday, noon of week 12 (April 9, noon)
Code (and resubmission of design) due in Blackboard Friday, noon of final's week (May 14, noon)

Office Supply Order System - A supply order consists of a unique order ID (numeric, system generated), a customer ID(alpha), a product ID(alphanumeric), a date of the order, an amount. A RepeatedOrder also has a period (the number of days between repeats), and an end Date for the repeating.

The Office Supply Order System needs to allow an Order Manager to manage a set of supply orders. Create a UserInterface class for your Office Supply Order System. This should be the project "void main" that allows the user these options:

1. specify a file name of orders to load
2. specify the information to add an order or repeated order
3. specify an orderID to delete
4. specify a CustomerID to display a list of orders for that customer in increasing date sorted order
5. calculate and output an order inventory report based on the current supply orders. The output file should be a comma-delimited .csv file, sorted by productID, year, month.

Project Inventory: Sort the orders by productID. If two orders have the same productID, look at the year. If the year is the same, then look at the month.

productID - alphabetical order
Split, then sort it in an array, then compare every index.

If two orders have the same productID

Two users order the same product ->

Use compareTo (-1, 0 is the same)

If productID, doesn't work

Each non "void main" class for the Office Supply Order System needs its own individual test program. Each non "void main" class for the Office Supply Order System needs its own individual exception class and should throw exceptions for any invalid action. Other classes need to catch these exceptions where appropriate.

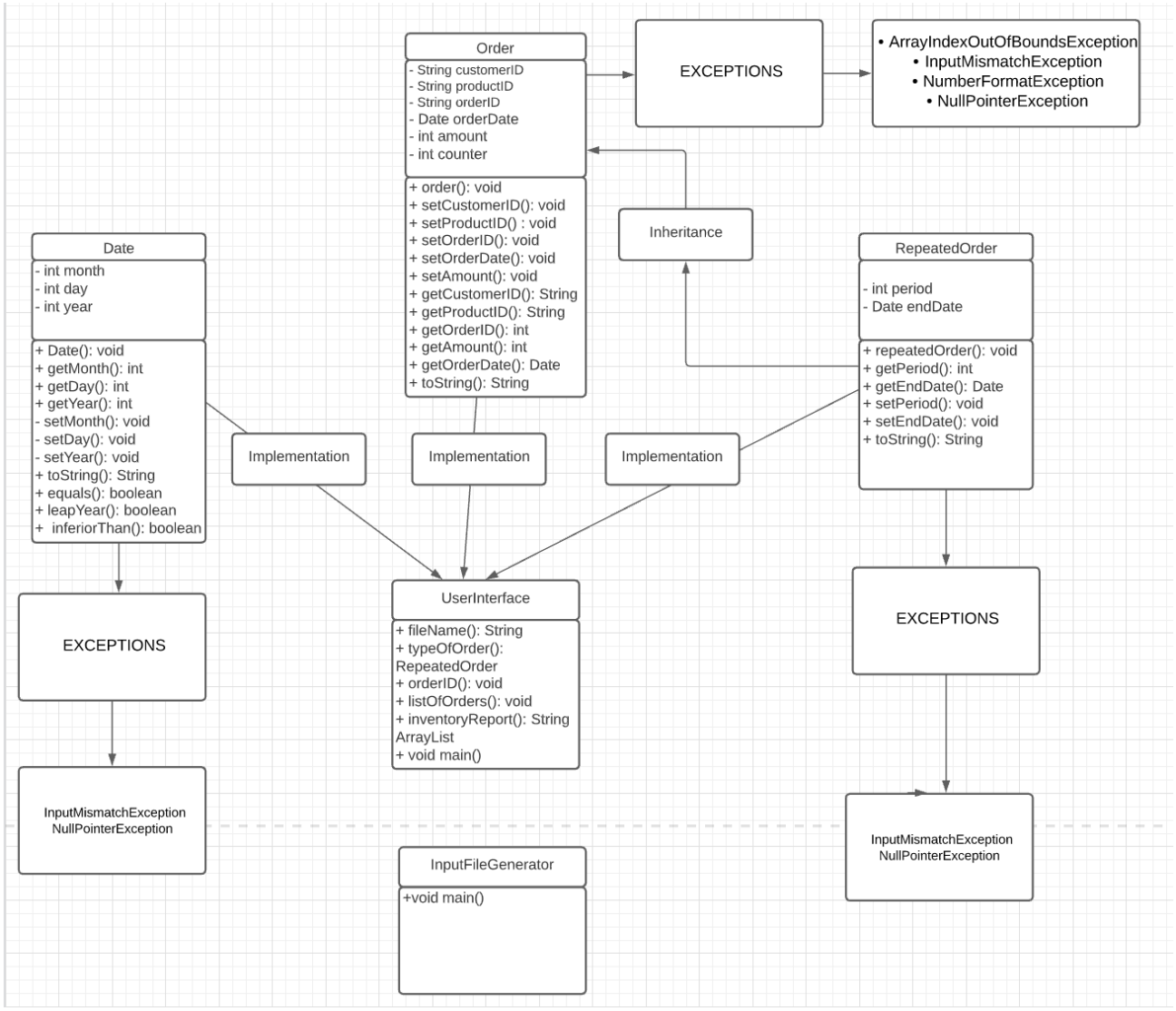
See the sample input file orderssmall.txt, with these fields, comma-delimited. O or R (onetime or repeated order), customerID, productID, orderDate, orderAmount, and if a repeated order, period and endDate. You will need to write an input file generator that will generate random input files. This should be its own "void main". Assume:

- customerIDs comes from the current NYSE Company list NYSEcompanylist.txt
- productIDs are all a single capital letter followed by a single numeric digit
- orderDates are all in 2021
- orderAmounts are all integers less than 100
- repeatedOrderPeriods are all 1 to 30 days
- repeatedOrderEndDates all in 2021

Design Requirements - Create the following for the above requirements (see UML_Design_Reference.pdf):

1. (2 points) Create a list of nouns and noun phrases from the above Requirements Document.
    a. Supply Order
        i. Unique Order ID (numeric, system generated)
        ii. Customer ID (alpha)
        iii. Product ID (alphanumeric)
        iv. Date of the Order
        v. Amount
    b. RepeatedOrder
        i. Period (number of days between repeats)
        ii. End Date: repeating orders
    c. Office Supply Order System - Order Manager, set of supply orders.
    d. "Void main": UserInterface class , Office Supply Order System
        i. File name
        ii. Information - Order or RepeatedOrder
        iii. OrderID - delete
        iv. CustomerID - orders, increasing data sorted order
        v. Order inventory report
            1. Output File:
                a. Comma-delimited .csv file
                b. Sorted by productID, year, month
    e. Non "Void main" class: Office Supply Order System, individual test program, individual exception class, throw exceptions
    f. Sample input file: O or R (onetime or repeated order), customerID, productID, orderDate, orderAmount, period and endDate (if a repeated order)
    g. Input file generator - generate random input files
    h. "Void main" Valid Input
        i. customerIDs - current NYSE Company list NYSEcompanylist.txt
        ii. productIDs - single capital letter followed by a single numeric digit
        iii. orderDates - 2021
        iv. orderAmounts - integers less than 100
        v. repeatedOrderPeriods - 1 to 30 days
        vi. repeatedOrderEndDates - in 2021

2. (2 points) Create Class Diagrams only for the nouns and noun phrases that have significance.



3. (3 points) Show the associations between the classes and any classes that are compositions of other classes.

Since the repeatedOrder class extends the Order class, there is inheritance.

The UserInterface class implements the methods from the other classes (Date, Order, repeatedOrder).

4. (5 points) Add Attributes to the Class Diagrams created earlier.
   a. customerID
   b. productID
   c. orderDate
   d. orderID
   e. orderAmount
   f. repeatedOrderPeriods
   g. repeatedOrderEndDate

5. (5 points) Add Operations to the Class Diagrams created earlier.
See class diagram

6. (3 points) Create test cases for each class.

Date Class:

Test Case Reason	Sample Data
Month >= 1 && Month <= 12	Month = 6
(Month == 4 || Month == 6 || Month == 11) && (Days >= 1 && Days <= 30)	Month = 6 && Days = 15
Month == 1 && (Days >= 1 && Days <= 31)	Month = 1 && Days = 16
Month == 2 && (Days >= 1 && Days <= 28)	Month = 2 && Days = 14
Year == 2021	Year = 2021
Date != null	Date = 05/43/2021

Order Class:

Test Case Reason	Sample Data
Amount > 0	Amount = 50
customerID != null	customerID = Orange Inc.
productID != null	productID = C2
orderDate != null	orderDate = 10/22/2021
Order != null	O = 1, C = Orange Inc., P = C2, D = 10/22/2021

Repeated Order:

Test Case Reason	Sample Data
Amount>0	Amount=50
customerID != null	customerID = Orange Inc.
productID != null	productID = C2
orderDate != null	orderDate = 10/22/2021

period>0	Period=7
endDate!=null	endDate= 6/8/2021
repeatedOrder != null	Orange Inc., C2, 10/22/2021, 7, 30, 12/8/2021

//O = orderID

FOR REPEATED ORDER:
1- constructor is the same as Order in addition to int period and Date endDate as it extends the Order class;
2- Period OK, endDate OK;
3- Period OK, endDate null;

4- Period<0, endDate OK;

5- setPeriod arg OK;
6- setPeriod arg<0;

7- setEndDate arg OK;
8- setEndDate arg null;

FOR ORDER Class:

1- default constructor
2- non-default construct, all 5 args OK
3- non-default construct, all 5 args NOT OK
4- non-default construct, Amount OK, customerID OK, productID OK, date null;
5- non-default construct, Amount OK, customerID OK, productID null, date OK;
6- non-default construct, Amount OK, customerID null, productID OK, date OK;
non-default construct, Amount<0, customerID OK, productID OK, date OK;

7- setAmount, arg OK
8- setAmount, arg <0;

9- setOrderDate arg OK
10- setOrderDate null;

11- setcustomerID arg OK
12- setcustomerID null;

13- setproductID arg ok;
14- setproductID null;

Months with 30 days: - April (4), June(6), and November (11)
Months with 31 days: - January (1)
Months with 28 days: - February (2)
Months with 29 days (leapYear): - February (2)

Constructor/mutators/toString Tests

For Date Class:

1- default constructor
2- non-default construct, all three args OK
3- non-default construct, all three args NOT OK
4- non-default construct, day OK, month OK, year=2021;
5- non-default construct, day OK, month<0, year OK;
6- non-default construct, day >30 (MONTH WITH 30 DAYS), month OK, year OK;
7- non-default construct, day >31 (MONTH WITH 31 DAYS), month OK, year OK;
8- non-default construct, day >28 (FOR FEBRUARY), month OK, year OK;
9- non-default construct, day >29 (FOR FEBRUARY IN LEAP YEAR), month OK, year OK;
10- non-default construct, day OK, month<=0 , year OK
11- non-default construct, day OK, month>12 , year OK
12- setMonth, arg OK
13- setMonth, arg <=0;
14- setMonth>12;
15- setDate arg OK
16- setDate, day >30 (MONTH WITH 30 DAYS);
17- setDate, day >31 (MONTH WITH 31 DAYS);
18- setDate, day >28 (For February);
19- setDate, day>29 (Leap year);
20- setYear arg OK
21- setYear year=2021;
22- leapYear (int year) Tests
    - arg negative
    - null argument
    - arg OK
23- comparedate(Date endDate) Tests
    - null argument
    - argument OK
    - default argument
24- Equals (Date d) Tests
    - null argument
    - default argument

UserInterface:

1. Specify a file name of orders to load
It should be a CSV file containing an ArrayList of orders and repeated orders.

2. Specify the information to add an Order or repeatedOrder
When a particular order is placed, then we must check if it's CustomerID. If the CustomerID matches, we'll check the order date and add it to the specific index accordingly. We'll use the ArrayList.add() method.

3. Specify an orderID to delete
When a particular orderID is to be deleted, the indices will be checked using the ArrayList.delete() method to delete that particular index if necessary.

4. Specify a CustomerID to display a list of orders for that customer in increasing date sorted order.

The UserInterface will check if the user input (customerID) matches with the CustomerID from the database, the ArrayList.sort() method will be implemented in order to output the customer's order in increasing date sorted order.

5. Calculate and output an order inventory report based on the current supply orders. The output file should be a comma-delimited .csv file, sorted by productID, year, month. The .split(“,”) method will be used in order to assign all the comma separated values at different indices in array1. Here, the index zero of the array will be compared to the corresponding productID and arrange it in ascending order. If the user's productID matches the productID from array1, then the first index of the arrayList and call the .split(“/”) to assign day, month, and year to different indices in a different array2. Then, the second indices of the array2 elements will be compared and arranged in ascending order based on the year. If the years match, then the first index of array2 will be compared then arranged in the ascending order according to the month.