PGCert IT: Programming for Industry

Homework Task 03

Time estimate: **120 minutes**

Notes:

- There are **three questions** in this practice task. It is worth **2.5%** of your final grade.
- You will be receiving marks by demonstrating your efforts on attempting the question.
- The question format will be similar to the actual test.
- You will be provided with a Zip file containing an IntelliJ project which will serve as a starting point for the task.
- To submit your answers, Zip your project and submit the single Zip file to Canvas or Moodle at or before the due date.
- **Important:** Make sure to read all instructions carefully before attempting the question.

Question One: A Duck Counter

Ben has been observing the number of ducks and ducklings in different ponds for several years. We like to build a simple duck counter application that will process the information recorded by Ben in a given csv file. The application will print the first 10 records to the console. In addition, the application will also print the first 10 records sorted by the total number of ducks in descending order.

In this question, you are required to complete the instructions listed in steps 1 – 3. The classes are provided to you in the duckstats folder. You are required to create the following class:

- `DuckCounterException`

You are also required to modify the following class:

- `DuckCounterProcessor`

You may also modify the `DuckCounter` class such that it doesn't break the supplied code.

After completing these steps, the application should produce similar output to the example overleaf.

1. Create one customized exception class, which extends the `Exception` class. The class has a constructor for receiving a string parameter. This string will be passed to the constructor of the superclass. The name of the class should be `DuckCounterException`.

2. Modify the `DuckCounterProcessor` class to import the given CSV file, `ducks.csv`. The file contains a number of rows. Each row contains a date, along with the total number of ducks and ducklings observed on that date. Note that there are no headings in the file.

    a. Modify the `processFile()` method to read the given csv file and return the list of `DuckCounter` objects. Here is some pseudocode to help you to complete this method:

        - Declare and initialize a list of `DuckCounter` objects.

        - While you are going through each entry from the csv file, you will need to first split each entry from a string into a string array by the delimiter ",". Then, you use the elements from the string array as the parameters for creating a new `DuckCounter` object, and add the `DuckCounter` object to the list that you created at the beginning of the method.

          Hint: Use a `BufferedReader` to read the lines in the file!

- If any exception is caught, print the Exception object's error message.
- Return the list of `DuckCounter` objects at the end of the method.

b. Complete the `printFirstTenDuckCounts()` method to print the first 10 duck statistics to the console as shown in the example output. **Hint:** You can use the `DuckCounter` object's `getDate().toString()` method to get a nice string representation of the date.

3. Modify the `DuckCounterProcessor` class's `main()` method so that it prints the first 10 duck statistics in descending order of the total number of ducks. **Hint:** One solution requires a modification to the `DuckCounter` class.

```
Processing Duck Data
----------------------
Error in duck data!

=================
First 10 Duck Stats
=================
Date: 2007-05-23, ducks: 3, ducklings: 1
Date: 2007-05-24, ducks: 6, ducklings: 1
Date: 2007-05-25, ducks: 1, ducklings: 1
Date: 2007-05-26, ducks: 6, ducklings: 1
Date: 2007-05-27, ducks: 2, ducklings: 1
Date: 2007-05-29, ducks: 1, ducklings: 1
Date: 2007-05-30, ducks: 1, ducklings: 1
Date: 2007-06-04, ducks: 1, ducklings: 1
Date: 2007-06-23, ducks: 2, ducklings: 1
Date: 2007-06-24, ducks: 2, ducklings: 1
=================
First 10 Duck Stats in Descending Order
=================
Date: 2011-08-05, ducks: 1439, ducklings: 760
Date: 2011-06-28, ducks: 1329, ducklings: 568
Date: 2011-08-17, ducks: 1316, ducklings: 702
Date: 2011-08-23, ducks: 1310, ducklings: 662
Date: 2011-06-27, ducks: 1252, ducklings: 484
Date: 2012-11-27, ducks: 1250, ducklings: 780
Date: 2012-06-21, ducks: 1225, ducklings: 750
Date: 2013-02-07, ducks: 1220, ducklings: 756
Date: 2013-02-21, ducks: 1204, ducklings: 737
Date: 2013-01-12, ducks: 1193, ducklings: 779
=================
```

Question Two: Game Collector

We would like to build a simple application that processes game details from a given csv file. The application will count the number of games by genre. It will also provide a summary of the number of games per year. In addition, the application can export the list of games to a csv file.

In this question, you are required to complete the instructions listed in steps 1 – 3. Some classes are provided to you in the appropriate folder. You are required to create the following classes:

- `GameInFutureException`
- `InvalidYearException`
- `Genre`

You are also required to modify the following class:

- `GameCollector`

You may also modify the `Game` class such that it doesn't break the supplied code.

After completing these steps, the application should produce output similar to that given overleaf. It should additionally produce a CSV file containing contents similar to the following, when opened in Excel:

Madden NFL 2004	2016	Electronic Arts
Call of Duty: Advanced Warfare	2014	Activision
FIFA 15	2014	Electronic Arts
Pokemon Omega Ruby/Pokemon Alpha Sapphire	2014	Nintendo
Super Smash Bros. for Wii U and 3DS	2014	Nintendo
Grand Theft Auto V	2014	Take-Two Interactive
Call of Duty: Ghosts	2013	Activision
Call of Duty: Ghosts	2013	Activision
FIFA Soccer 14	2013	Electronic Arts
MineCraft	2013	Microsoft Game Studios

1. Create an *enum* called `Genre`. It should have the following set of constants: MISC, ACTION, RPG, SPORTS, PUZZLE, and OTHER.

2. Create two customized exception classes, each extending from `Exception`. Each class should have a constructor for receiving a string parameter. The classes should be called `GameInFutureException` and `InvalidYearException`.

3. Modify the GameCollector class to scan the given CSV file, print the first five games, find the number of games by genre, sort the list of games by year and publisher, export the list of games to a file, and print the number of games per year. To do this, perform the following steps:

a. Modify and complete the checkYear() method so that it converts a given string to an integer and returns the integer value. If the string is empty, the return value is 2016. If the integer is less than 1950, an InvalidYearException will be thrown, with the message “The game is too old!”. If the value is greater than 2016, a GameInFutureException will be thrown, with the message “The game is too new!”.

b. Modify and complete the getGenre() method so that it returns the genre of the game based on the given string. If the given string is “action”, the genre is ACTION. If the given string is “misc”, the genre is MISC. If the given string is “role-playing”, the genre is RPG. If the given string is “puzzle”, the genre is PUZZLE. If the given string is “sports”, the genre is SPORTS. Otherwise, the default genre is OTHER. Note that you will need to ignore the case for the given string.

c. Modify the processGameDetails() method appropriately to throw InavalidYearException and GameInFutureException.

d. Complete the processFile() method to read the given CSV file and store each entry into the games list. Each row in the CSV file represents a game, and includes that game’s id, name, platform, year, genre, and publisher. **Note:** The first line in the CSV file is a heading and should not be interpreted as a game. Here is some pseudocode for the method:

    - Initialise the games variable, which stores a list of Game object.
    - While you are going through each entry from the csv file, use the processGameDetails() method to create a new Game object. The method takes a string array as the parameter. You will need to first split each sentry from a string into a string array by the delimiter “,” and then pass the string array as the parameter to the processGameDetails() method. Add each Game object to the games list.
    - Catch all the exceptions in this method, including InavalidYearException and GameInFutureException. If any exception is caught, you print the exception object’s error message to the console.

e. Complete the printFirstFiveGames() method so that it prints the first 5 games to the console as shown on the screenshot. **Hint:** Make use of Game’s toString() method.

f. Complete the getNumberOfGamesByGenre() to count the number of games from the games list that belong to the given genre.

g. Complete the printTopTenSortedGames() method so that it prints the first 10 games in descending order of the year. If the year is the same, then you sort

the game in ascending order of the publisher. **Hint:** One solution requires a modification to the Game class.

h. Complete the exportSortedGames() method to export the game details to a file specified by the filePath parameter. If any exception is caught you print the exception object's error message. The content of the file contains only the name, the year, and the publisher for each game from the games list.

i. Complete the getCountOfGamesPerYear() method to count the number of games per year. Note that the method returns a map. The key of the map should be the year, and the value should be the number of games.

```
The first 5 games
==================
The game "Wii Sports" (id:1) was released by "Nintendo" in 2006
The game "Super Mario Bros." (id:2) was released by "Nintendo" in 1985
The game "Mario Kart Wii" (id:3) was released by "Nintendo" in 2008
The game "Wii Sports Resort" (id:4) was released by "Nintendo" in 2009
The game "Pok?mon Red / Green / Blue Version" (id:5) was released by "Nintendo" in 1996
==================
Number of Action games in the collection: 34
Number of Puzzle games in the collection: 4
Number of Role-Playing games in the collection: 31
Number of Sports games in the collection: 15
==================
The top 10 games by year and publisher
==================
The game "Madden NFL 2004" (id:171) was released by "Electronic Arts" in 2016
The game "Call of Duty: Advanced Warfare" (id:130) was released by "Activision" in 2014
The game "FIFA 15" (id:147) was released by "Electronic Arts" in 2014
The game "Pokemon Omega Ruby/Pokemon Alpha Sapphire" (id:83) was released by "Nintendo" in 2014
The game "Super Smash Bros. for Wii U and 3DS" (id:143) was released by "Nintendo" in 2014
The game "Grand Theft Auto V" (id:132) was released by "Take-Two Interactive" in 2014
The game "Call of Duty: Ghosts" (id:64) was released by "Activision" in 2013
The game "Call of Duty: Ghosts" (id:65) was released by "Activision" in 2013
The game "FIFA Soccer 14" (id:109) was released by "Electronic Arts" in 2013
The game "MineCraft" (id:103) was released by "Microsoft Game Studios" in 2013
==================
Number of games per year
==================
Games:1     Year: 1982
Games:1     Year: 1984
Games:1     Year: 1985
Games:1     Year: 1986
Games:3     Year: 1988
Games:3     Year: 1989
Games:1     Year: 1990
Games:5     Year: 1992
Games:1     Year: 1993
Games:3     Year: 1994
Games:1     Year: 1995
Games:7     Year: 1996
Games:6     Year: 1997
Games:7     Year: 1998
Games:11    Year: 1999
Games:3     Year: 2000
Games:10    Year: 2001
Games:5     Year: 2002
Games:5     Year: 2003
Games:9     Year: 2004
Games:8     Year: 2005
Games:10    Year: 2006
Games:15    Year: 2007
Games:12    Year: 2008
Games:16    Year: 2009
Games:17    Year: 2010
Games:13    Year: 2011
Games:11    Year: 2012
Games:8     Year: 2013
Games:5     Year: 2014
Games:1     Year: 2016
```

Question Three: Simple Phone Book

We like to build a simple phone book application that stores the names and phone numbers in a collection. The phone book application creates a SimplePhoneBook object and calls the instance methods of that object to do things such as adding new contacts, displaying the phone number for a particular person, and finding the person with a particular phone number.

In this question, you are required to complete the SimplePhoneBook class. The classes are provided to you in the simplephonebook folder. You may create additional classes if needed.

After completing the SimplePhoneBook class, the application should produce similar output as the following example:

```
Welcome to the Simple Phone Book App
====================================
Please enter the name you like to add: Doraemon
Please enter the number for that person: 1234567
------------------------------------
The person Kate does not exist in the phone book.
------------------------------------
Current number for George is 4197108.
------------------------------------
There are 7 entries in the phone book.
------------------------------------
The person with the number 1234567 is: Doraemon.
```

In the SimplePhoneBook class, you must have the following methods:

- **addMultipleContacts():** This method takes a Map object as the parameter, which contains multiple key / value pairs, where the keys are people's names and the values are their phone numbers. The method should add each entry of the given Map to the class's own collection.

- **addNewContact():** This method takes a name and a phone number as the parameters of the method. It adds the name and the phone number to the class's own collection. The method also prints a message for the newly added contact to the console, e.g. "Number for Doraemon is 1234567". If the name already exists in the collection, the method will print a message to the console, e.g. "Current number for Doraemon is 7654321", before the message for the newly added contact.

- **printNumberFor():** This method takes a name as the parameter, and prints the number for the given name to the console. An example message is "Current number for George is 4197108". If the name does not exist in the collection, a message with the given name like "The person Kate does not exist in the phone book" will be printed.

- **printTotalNumberOfEntries():** This method prints the number of entries in the class's collection to the console. An example message is "There are 7 entries in the phone book".

- **printNameByPhoneNumber():** This method takes a phone number as the parameter, and prints the name for the given number to the console. An example message is "The person with the number 1234567 is: Doraemon". If the number does not exist in the collection, a message with the given number like "Cannot find person in the phone book with the number 7654321" will be printed.

Hint: A constructor for the SimplePhoneBook class may be useful. You may use the constructor to initialize the collection for storing names and phone numbers.