Concordia University Comp 249 – Summer 2023
Object-Oriented Programming II
Assignment 2
Due 11:59 PM – Friday, July 14, 2023

1 Purpose

The purpose of this assignment is to allow you to practice Exception Handling and File I/O, as well as other previously covered object-oriented concepts.

2 Introduction

This assignment is supposed to help you reorganize data related to sports teams. Over the years you were collecting in a text files information about sports team (sport, team name, record of the team, the year of the games). You decide to split the file into different files one for each sport (Hokey, Football and Basketball). Your work will be divided into three parts:

1- Part 1: Reading files and organizing sports files:
   You need to read a text file having a list of files holding information about the sports' teams over the year (one file per year) and extracting from the different input files the information about each sport and creation of three sports files and one file holding the incorrect information collected from the original text files.
2- Part 2: Reading Sports Files and Serializing Team Objects.
   Read the sports files generated in Part 1 for each sport. Create an array of team objects based on the information in the sports files. Serialize the resulting team objects into a binary file for each sport. Handle exceptions for file write errors.
3- Part 3: Interactive Navigation and Data Analysis.
   Implement an interactive menu for the user to navigate through the team objects in the arrays. Initially, set the current object in the array to the first object at index 0. Allow the user to move up or down relative to the current object in the array. Provide options for the user to choose:
   a) Choose a specific sport and display information about the team with the highest score, including the team name and the year of achievement.
   b) Ask for the lowest score data, including the team name and the year of achievement.

   N.B. Handle user input validation to ensure correct navigation and option selection.

3 Your Assignment

Write a driver program whose main() method implements the requirements of this assignment in three sequentially dependent parts described in the following pages.

```
public static void main(String[] args)
{
   do_part1();        // validating syntax, partition book records based on genre.
   do_part2();        // validating semantics, read the genre files each into arrays of Book objects,
                      // then serialize the arrays of Book objects each into binary files.
   do_part3();        // reading the binary files, deserialize the array objects in each file, and
}                     // then provide an interactive program to allow the user to navigate the arrays.
```

4 CSV-Formatted Input Files

This assignment requires input data from one or more text files, including a file called Part1_input_file_names.txt that stores the names of the required input files, one per line, starting at the second line. The first line stores the number, say n, of the file names listed below it (3 in the example below).

```
part1 input file names.txt
3
games2010.csv
games2011.csv
games2019.csv
```

The names of the files listed in Part1_input_file_names.txt may be any valid file name, and the files themselves each may or may not exist, and if they do exist, they may or may not be empty. If a file does not exist, you will simply display an error message and move on to the next input file, if any. Each line of an input file represents a team performance detail record. Each record terminates by a new line character and has information about the name of the team, the type of the sport, the year, the team record using the format #won-#loss and a flag championship if the team won the championship that year or not. The data fields themselves are separated by a comma character, a textual format called "CSV" (comma separated values).
TeamName, SportType, year, Team Record, championship
A team record may contain either syntax or semantic errors:
Syntax errors may occur when extracting the data fields of a CSV-formatted input line. Syntax errors are related to the number of fields in each record.
An invalid sport is considered a syntax error, an invalid team record is also a syntax error, an incorrect number of fields is also a syntax error.
Semantic errors may occur when validating the values in the record: the year, the team record values, the sport.

Example of Syntax errors:

```
Canadiens, Hokey, 2001, , N                    //missing field
Avalanche, Hokey, 2001, 52-16                  // missing field
Stampeders, Football, 2001, 3-2, Y, y          // too many fields
```

Example of semantic errors:

```
Canadiens, Hokey, 2001, 36 31, N // incorrect team record
Avalanche, Hokey, 1111, 52-16, Y // incorrect year
Avalanche, ABC, 1111, 52-16, Y // invalid sport
```

Example of correct records

```
Canadiens, Hokey, 2001, 36-31, N
Avalanche, Hokey, 2001, 52-16, Y
Stampeders, Football, 2001, 3-2, Y
Lions, Football, 2011, 11-7, Y
1 Lions, Football, 2015, 7-11, N
```

5 Part 1 of 3

Write a method called do_part1() that will read team records from a number of CSV-formatted text files, checking only for syntax errors.

part1 input file names.txt

3
games2010.csv
games2011.csv
games2019.csv

Text Files produced in Part 1

Hokey.csv
Football.csv
Basketball.csv
syntax_error_file.txt

Preparing the input files for Part 2, Part 1 will output four files, three CSV-formatted text files and a regular text file, whose names are listed in the above at right.
Since the names and number of the output files and sport's type may vary depending on the input files, you would ideally create a text file, similar to Part1_input_file_names.txt in format, and enter the names and number of the output CSV-files; then, you would use that file as input in Part 2.

Processing the records in each of the CSV-formatted input files (listed in Part1_input_file_names.txt), you will encounter records that are either syntactically valid or not.
If a record is syntactically valid, then you will write that record to a sport-based CSV-formatted output file (listed above). You will need to keep track of the size of (the number of teams in) each of the CSV-formatted output files so as to minimize the number of times a single file is opened and closed.

If you encounter a syntactically invalid team record, then you will throw an exception corresponding to the syntax error detected, reporting (a) the error, (b) the record, and (c) the name of the file in which the record appears. To accommodate this case, write four checked exception classes called TooManyFieldsException, TooFewFieldsException, MissingFieldException, and UnknownSportException.
Here is an example of the contents of syntax_error_file.txt :

syntax_error_file. txt

```
1 syntax error in file: games2001.csv
2 ====================
3 Error: record field
4 Record: Canadiens, Hokey, 2001, , N
5
6 syntax error in file: games2001.csv
7 ====================
8 Error: missing championship field
9 Record: Avalanche, Hokey, 2001, 52-16
```

6 Part 2 of 3

Recall that except for the file syntax_error_file.txt, all the other files listed below are sport-based CSV-formatted text files containing syntactically valid team records.
This part checks each of the syntactically valid team records for semantic errors, setting out to serialize both syntactically and semantically valid team records into binary files whose names appear below at the right. (Again, you may hard-code the names of the binary files in your program.)

Text Files produced in Part 1
Hokey.csv
Football.csv
Basketball.csv
syntax_error_file.txt

Binary Files produced in Part 2
Hokey.csv.ser
Football.csv.ser
Basketball.csv.ser

Write a method called do_part2() that reads the sport-based CSV-formatted input text files produced in Part 1, one file at a time, creating an array of valid team objects out of all the semantically valid team records in each input file. A line in an input file, which is already valid syntactically, is processed as follows depending on whether the line contains a semantic error:

a) The line contains a semantic error rendering it invalid:

Throw an exception of the type corresponding to the semantic error detected, reporting the error into an error file named semantic_error_file.txt using the same format as that appearing in the file syntax_error_file.txt.

To accommodate this case, write three checked exception classes BadRecordException, BadYearException, and BadSportException.

b) The line contains a semantically valid book record:

Create a team object based on the data field values of the team record and then store the object in the array of team objects set up for the file containing the valid record.

To accommodate this case, write a simple Team class that implements Serializable and has five instance variables: name, sport, year, score and championship; initialize the instance variable in a team constructor taking five parameters corresponding to the five instance variables. It should override the equals() and toString() methods, and for each instance variable, define a pair of corresponding getter and setter methods. Feel free to introduce your own methods to facilitate the implementation of the operations involved in this assignment.

Once you have processed all the records in a file, say abc.csv, you will then serialize the resulting array of team objects into a binary file named abc.csv.ser and then move on to processing the next input file.

7 Part 3 of 3

Write a method called do_part3() that will open each of the three binary files produced in Part 2, deserializing the object in each binary file into an array of team objects. (You may hard-code the names of the binary files in your program if you wish.) You will then write an interactive code to navigate the objects in any of the arrays of team objects, moving up or down relative to the current object in the array, which is initially set to the first object in the array at index 0.

Binary Files produced in Part 2
Hokey.csv.ser
Football.csv.ser
Basketball.csv.ser

The position of the current object is adjusted according to whether the range of objects being viewed (displayed) are above or below the current object; the adjustment process is detailed on the next page. Your interactive code must repeatedly display the following menu and perform the selected menu item until the user enters the letter x or X on the keyboard:

```
----------------------------
Main Menu
----------------------------
v View the selected file: Hokey.csv.ser (3 records)
s Select a file to view
x Exit
----------------------------
Enter Your Choice: s
```

Note that ”selected file” in the menu effectively refers to the array deserialized from that file. Option v will allow the user to navigate the currently selected file, initially, any one of the arrays.
Option s will prompt the user to select a file through the following menu:

```
----------------------------
File Sub-Menu
----------------------------
1 Hokey.csv.ser (8 records)
2 Football.csv.ser (3 records)
3 Basketball.csv.ser (5 records)
4 Exit
----------------------------
Enter Your Choice: 2
```

The main menu will now display again:

```
----------------------------
Main Menu
----------------------------
v View the selected file: Football.csv.ser (1 records)
s Select a file to view
x Exit
----------------------------
Enter Your Choice:
Option v opens as follows:
Enter Your Choice: v
viewing: Football.csv.ser (5 records)
```

The viewing commands are only integers, say number n, each specifying a range of at most n consecutive records to be displayed, unless n = 0.

- If n = 0, then the viewing session ends and control will display the main menu again.
- Whether n < 0 or n > 0,
  - the current object in the array is always displayed.
    - Hence, entering +1 or −1 will display only the current record.

- If n > 0, then the current object and the (n − 1) objects below it, if any, are displayed.
  - If there are not (n − 1) records below the current object, then after displaying the last object in the array, the message EOF has been reached is displayed.

- The last record in the displayed range will always become the new current object.

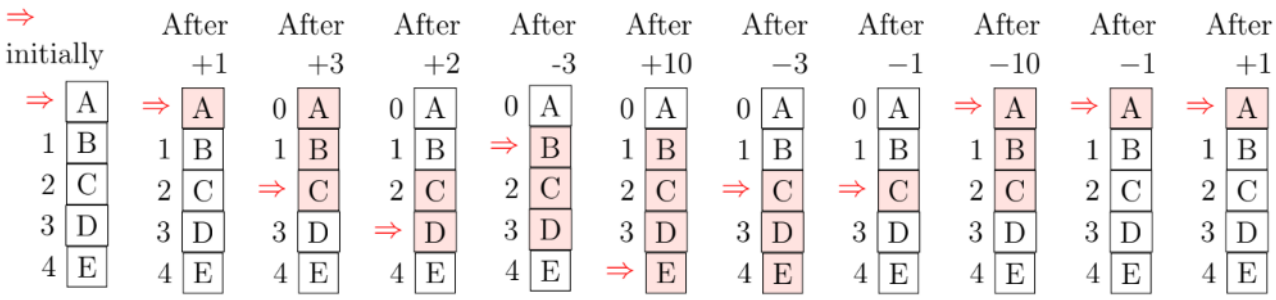
For example, if the current object is at the index, say 17, then for n = +3, the current object at index 17 and the (n−1) = (3−1) = 2 objects below it at indexes 18, 19, a total of 3 objects, are displayed and the current object will be located at index 19, the last object displayed in the range 17-19.

- If n < 0, then the current object and the (|n| − 1) objects above it, if any, are displayed.
    - If there are not (|n| − 1) objects above the current object, then before displaying the first object in the array, the message BOF has been reached is displayed.

  The first object in the displayed range will always become the new current object.

For example, if the current object is at the index, say 17, then for n = −3, the current object at index 17 and the (|n| − 1) = (|−3| − 1) = 3 − 1 = 2 objects above it at indexes 15, 16, a total of 3 objects, are displayed and the current object will be located at index 15, the first object displayed in the range 15-17.

For another example, consider the first array below at left where ⇒ points to the current object and note that the cells displayed in color represent the displayed objects .



8 Requirements

a. You may not use an object of a class in the Java Collections Framework, such as LinkedList, HashMap, TreeMap, HashSet, TreeSet, etc.. Nor may you use any external libraries or existing software to produce what is required.
b. For processing of input book records, you may use String's split method and/or the StringTokenizer class. For array processing, you may use the Java's Arrays class.
c. Feel free to introduce and implement classes of your choice to facilitate the implementation of the operations involved in this assignment.
d. You should minimize opening and closing the files.
e. Your program must work for any input files. The CSV files provided with this assignment are only one possible versions, and must not be considered as the general case when writing your code. You may rename all .csv files to .csv.txt files if you wish.

9 General Guidelines When Writing Programs

Include the following comments at the top of your source codes.

```
1 // ----------------------------------------------------
2 // Assignment (include number)
3 // Question: (include question/part number, if applicable)
4 // Written by: (include your name and student ID)
5 // ----------------------------------------------------
```

In a comment, give a general explanation of what your program does. As the programming questions get more complex, the explanations will get lengthier.
Include comments in your program describing the main steps in your program.
Display a welcome message which includes your name(s).
Display clear prompts for users when you are expecting the user to enter data from the keyboard.
All output should be displayed with clear messages and in an easy-to-read format.
End your program with a closing message so that the user knows that the program has terminated.

10 JavaDoc Documentation

Documentation for your program must be written in javaDoc. In addition, the following information must appear at the top of each file:

- Name(s) and ID(s) (include full names and IDs)
- COMP249
- Assignment # (include the assignment number)
- Due Date (include the due date for this assignment)

11 What to submit

Your submission for this assignment must include:

- All Java files related to this assignment,
- all CSV files, both input and output, and
- the two error files

12 Assignment Submission Guidelines

- For this assignment, you are allowed to work individually, or in a group of a maximum of 2 students (i.e., you and one other student).
- Only electronic submissions will be accepted. Zip together the source codes.
- Naming convention for zip file: Create one zip file, containing all source files and produced

documentations for your assignment using the following naming convention: The zip file should be called a#_StudentName_StudentID, where # is assignment number and StudentName and StudentID is your name and ID number respectively. Use your "official" name only.

- no abbreviations or nick names; capitalize the usual "last" name. Inappropriate submissions will be heavily penalized. For example, for the first assignment, student 12345678 would submit a zip file named like: a1_Mike-Simon_123456.zip. if working in a group, the name should look like: a1_Mike-Simon_12345678-AND-Linda-Jackson_98765432.zip.
- Submit only ONE version of an assignment. If more than one version is submitted the first one will be graded, and all others will be disregarded.
- If working in a team, only one of the members can upload the assignment. Do NOT upload the file for each of the members!

=⇒ Important: Following your submission, a demo is required (please refer to the course outline for full details). The marker will inform you about the demo times. Please notice that failing to demo your assignment will result in zero mark regardless of your submission.

13 Evaluation Criteria for Assignment 2 (10 points)

Total	**10 pts**
JavaDoc documentations	1 pt
Producing proper CSV output files in Part 1	2 pt
Producing proper binary files in Part 2	2 pts
Implementing the interactive Part 3	2 pts
Generating and Handling Exceptions	2 pt
General Quality of the Assignment	1 pt