CSE 8B: Introduction to Programming and Computational Problem Solving - 2

Assignment 7

Exception Handling and Text I/O

Due: Wednesday, November 16, 11:59 PM

Learning Goals:

- Understanding exception handling in practice
- Using text file I/O to read and write to files

NOTE: This assignment must be completed INDIVIDUALLY. Pair programming is NOT allowed for this assignment.

Your grade will be determined by your most recent submission. If you submit to Gradescope after the deadline, it will be marked late and the late penalty will apply regardless of whether you had past submissions before the deadline.

If your code does not compile on Gradescope, you will receive an automatic zero on the assignment.

Coding Style (10 points)

For this programming assignment, we will be enforcing the CSE 8B Coding Style Guidelines. These guidelines can also be found on Canvas. Please ensure to have COMPLETE file headers, class headers, and method headers, to use descriptive variable names and proper indentation, and to avoid using magic numbers.

Part 0: Getting started (0 points)

1. Make sure there is no problem with your Java software development environment. If there is any, then review Assignment 1, or come to the office/lab hours before you start Assignment 7.
2. First, navigate to the `cse8b` folder that you have created in Assignment 1 and create a new folder titled `assignment7`
3. Download the starter code. You can download the starter code from Piazza → Resources → Homework → `assignment7.zip`. The starter code should contain the file `TicTacToe.java` along with three exception class files named `InvalidCharacterException.java`, `InvalidGridIndexException.java` and `InvalidGridDimensionException.java`. It should also contain a text file named `readBoard.txt`. Place the starter code within the `assignment7` folder that you have just created.
4. Compile the starter code within the `assignment7` folder. You can compile all files using the single command `javac *.java` and you should get a series of compiler errors since you have not implemented the classes yet. The objective of this assignment is to get the classes working by implementing the class methods and testing them.
5. Next, please make the `readOnlyFile` inside `assignment7` to be read-only. Here's what you should do:
    a. If you're using Linux (macOS, Ubuntu, etc), go to your `assignment7` directory. Run the following command.

    ```
    $ chmod 444 readOnlyFile
    ```

    b. If you're using Windows, go to your `assignment7` folder. Right click the `readOnlyFile`. Click properties. Then check the Read-only box.

Part 1: Implement InvalidCharacterException class (5 points)

You will be implementing the class `InvalidCharacterException`, which extends the Java class `Exception`. An instance of `InvalidCharacterException` is thrown when the input file contains an invalid character to enter into the grid (explained in the later part of the assignment). You only need to implement the no-arg constructor by calling the superclass constructor `Exception(String message)` with the constant string `EXCEPT_MSG` provided to you in the starter code.

***Food for thought**: Is `InvalidCharacterException` a checked exception or unchecked exception?*

Part 2: Implement InvalidGridIndexException class (5 points)

You will be implementing the class `InvalidGridIndexException`, which extends the Java class `Exception`. An instance of `InvalidGridIndexException` is thrown when the input file contains an invalid index to enter into the grid (explained in the later part of the assignment). You only need to implement the no-arg constructor by calling the superclass constructor `Exception(String message)` with the constant string `EXCEPT_MSG` provided to you in the starter code .

***Food for thought**: Is `InvalidGridIndexException` a checked exception or unchecked exception?*

Part 3: Implement InvalidGridDimensionException class (5 points)

You will be implementing the class `InvalidGridDimensionException`, which extends the Java class `Exception`. An instance of `InvalidGridDimensionException` is thrown when the dimensions of the input grid does not match `NUM_ROWS` and `NUM_COLS` (explained in the later part of the assignment). You only need to implement the no-arg constructor by calling the superclass constructor `Exception(String message)` with the constant string `EXCEPT_MSG` provided to you in the starter code.

***Food for thought**: Is `InvalidGridDimensionException` a checked exception or unchecked exception?*

Part 4: Implement TicTacToe class (65 points)

In `TicTacToe.java`, you will implement functions to check the status of a Tic-Tac-Toe game. You will be implementing **2 constructors and 5 other methods**.

Each `TicTacToe` object, which is an instance of the `TicTacToe` class, contains the field `private char[][] grid` which is a Tic-Tac-Toe grid with `NUM_ROWS` and `NUM_COLS` (all are provided in the starter code; do not change any of these).

The `TicTacToe` class contains the following member methods:

Note: (Please implement and complete the method body but do not change the method headers)

1. **`public TicTacToe()`**

This is the no-arg constructor of the `TicTacToe` class. The constructor creates a default game `grid` with `NUM_ROWS` and `NUM_COLS`, and initializes each element with `E_CHAR` (stands for empty character).

2. **`public TicTacToe(String fileName)`**
   This is another constructor of the `TicTacToe` class. This constructor creates a `grid` with `NUM_ROWS` and `NUM_COLS`, and sets the grid by calling the `readGrid` method (explained later).

3. **`public void readGrid(String fileName)`**
   This method must read the grid's initial state from a file and parse the grid to a 2D character array, which is used to initialize the `grid` member variable. The method takes an input argument `fileName` which is the name of the file containing the grid values we want to read. (*Hint*: *use the* `Scanner` *class*).

   Following is the format of the text contained in `fileName` (you can create multiple files of the same format when testing your code).
   - The file must contain 10 lines. The first 9 lines will contain the grid elements. The last line is simply an empty new line.
   - Each line (in the first 9 lines) will have a row index, a column index, and the character. Each value is separated by a *single* space character. From each line, you will read the row, column and the grid character and store it at `grid[row][column]`, like below:

```
0 0 X
```

The above row means the following:

```
grid[0][0] = 'X';
```

Below is a sample of an entire file content:

```
0 0 O
0 1 X
0 2 E
1 0 O
1 1 O
1 2 X
2 0 X
2 1 E
2 2 O
\n
```

This should result in a `grid` that looks like this:

O	X	E
O	O	X
X	E	O

As mentioned above, `'E'` (defined as `E_CHAR` in starter code) is an empty character.

While reading `fileName`, there are multiple exceptions that may occur:

1. `IOException` - this exception is thrown when the file you are trying to read does not exist.

   Print a message while handling the exception using the following method.

   `exception.getMessage()`

   Note: This is the only exception you will handle inside the method. Other potential exceptions thrown by this method must be handled outside this method (e.g. `unitTests()`). DO NOT re-throw the exception)

2. `InvalidGridIndexException` - the `readGrid` method must throw this exception when one of the lines in the file contains an index that is outside the grid dimension `NUM_ROWS` and `NUM_COLUMS`.

   For example, in the starter code `NUM_ROWS` and `NUM_COLS` are assigned to 3. Therefore, the valid indices for rows are `{0, 1, 2}` and the valid indices for columns are `{0, 1, 2}`.

   Now, let's assume that the input file contains the following line:

   `2 3 X`

   This line indicates that you store `grid[2][3] = 'X'` which is an invalid assignment. You will throw `InvalidGridIndexException` for these cases.

3. `InvalidCharacterException` - the `readGrid` must throw this exception when one of the lines in the file contains a character that is not `X_CHAR`, `O_CHAR`, or `E_CHAR`.

   For example, let's assume that the input file contains the following line:

   `1 2 R`

This line indicates that you store `grid[1][2] = 'R'`. But the only valid characters the grid can have are `X_CHAR`, `O_CHAR`, or `E_CHAR`. You will throw `InvalidCharacterException` for these cases.

You can assume that as long as there is no exception, the file is valid for you to initialize the 2D array. Remember to close the `Scanner` object that you created to read the input file after the reading is finished.

(Hint: *try-with-resources statements might/might not be helpful, depending on whether you want to close resources explicitly.)*

4. **`public void writeGrid(String fileName)`**

This method must write the Tic-Tac-Toe grid contained in the `grid` variable to the file specified by the `fileName` argument. (***Hint**: Use the* `PrintWriter` *class*)

The output file must follow the same format as the input file:

1. It should contain 10 lines.
2. The first 9 lines should contain the grid elements for each position separated by a *single* space character. (Each grid element should be in a new line.)
3. The 10th line is an empty new line.
4. (i.e. there will be `NUM_ROWS` line in the file)
5. No extra spaces in any line.
6. You will lose points if your output file's format does not meet the requirements.

You will need to handle `IOException`. For `PrintWriter`, a new file will be created if the file specified by `fileName` does not exist. However, you still need to handle the `IOException` to avoid other cases. For example, throw this exception if the output file is a read-only file.

Print a message while handling the exception using the following method.

```
exception.getMessage()
```

Note: This is the only exception you will handle inside the method. Other potential exceptions thrown by this method must be handled outside this method (e.g. `unitTests()`). DO NOT re-throw the exception)

You can assume that as long as there is no exception, the grid state is valid for you to write to the file. Close the `PrintWriter` object that you created for writing to the output file after the writing is finished.

5. **`public void getGrid()`**

This method is an accessor for the member variable `grid`. Simply return the Tic-Tac-Toe grid in this method.

6. **`public void setGrid(char[][] grid)`**

This method is a mutator for the member variable `grid`. While setting the member variable `grid` with the parameter values, there are multiple exceptions that you need to check for while setting the grid.

1. `InvalidGridDimensionException` - the `setGrid` method must throw this exception when the input parameter has different dimensions than `NUM_ROWS` and `NUM_COLS`.

   For example, let's assume that the following argument is passed to the method:

   ```
   {{'X', 'O', 'X', 'E'},
    {'O', 'O', 'E', 'X'},
    {'E', 'E', 'X', 'O'}}
   ```

   This grid has 3 rows and 4 columns, while according to the values in the starter code `NUM_ROWS` and `NUM_COLS` are both assigned to 3. You will throw `InvalidGridDimensionException` in this case.

4. `InvalidCharacterException` - the `setGrid` method must throw this exception when one of the lines in the file contains a character that is not `X_CHAR`, `O_CHAR`, or `E_CHAR`.

   For example, let's assume that the input file contains the following line:

   ```
   1 2 R
   ```

   This line indicates that you store `grid[1][2] = 'R'`. But the only valid characters the grid can have are `X_CHAR`, `O_CHAR`, or `E_CHAR`. You will throw `InvalidCharacterException` for these cases.

7. **`public String checkWinner()`**

The method is used to check if `'X'` has won the game, `'O'` has won the game, or if the game is a draw.

To determine which string constant to return, there are four different game win conditions you need to check in `checkWinner`:

- *Row win condition* - There are three `'X'` characters or three `'O'` characters in the current row. Return either `X_WIN_MESSAGE` or `O_WIN_MESSAGE` (based on if it's `'X'` or `'O'`).

- *Column win condition* - There are three `'X'` characters or three `'O'` characters in the current column. Return either `X_WIN_MESSAGE` or `O_WIN_MESSAGE`.
- *Top-left to Bottom-right diagonal win condition* - There are three `'X'` characters or three `'O'` characters in the diagonal that starts top-left and ends bottom-right. Return either `X_WIN_MESSAGE` or `O_WIN_MESSAGE`.
- *Bottom-left to Top-right diagonal win condition* - There are three `'X'` characters or three `'O'` characters in the diagonal that starts bottom-left and ends top-right. Return either `X_WIN_MESSAGE` or `O_WIN_MESSAGE`.

If none of the previous conditions are satisfied, return `DRAW_MESSAGE`.

Be sure to compile your code often, so that you can catch compile errors early on! Recall, to compile multiple Java files, use:
`> javac *.java`

Part 2: Compile, Run and UnitTest Your Code (10 points)

Just like in previous assignments, **in this part of the assignment, you need to implement your own test cases in the method called** `unitTests` **in the** `TicTacToe.java` **class.**

You are encouraged to create as many test cases as you think to be necessary to cover all the edge cases. The `unitTests` method must return `true` only when all the test cases are passed. Otherwise, it must return `false`.

To get full credit for this section, you must create at least four test cases that cover different situations. In other words, you will need to create at least **four** tests that test `getGrid()`, `setGrid(char[][] grid)`, `checkWinner()`, `readGrid()`, `writeGrid()`, and the constructors `TicTacToe()` and `TicTacToe(String fileName)`. Remember to test for all possible edge cases - such as writing the grid into a read only file (the `writeGrid()` method should throw an exception in this case).

Each of your test cases should handle the exceptions that each method may throw appropriately. While handling each of these exceptions, print a message while handling the exception using the following method.

`exception.getMessage()`

If a test is not passing, try temporarily printing the result of your method(s) and comparing them to the expected output.

You can compile all the files present in the starter code and run your unit tests from `main()` using the following commands: (Make sure you are in the correct directory, else navigate to the starter code using cd)

```
> javac *.java
> java TicTacToe
```

The first command `javac *.java` compiles all the files in the folder with a `.java` extension, which is what is required. After the `.java` files compile and `.class` files are generated, run the `main()` in `TicTacToe` using `java TicTacToe`. The below screenshot shows the same:

```
$ javac *.java
$ java TicTacToe
All unit tests passed.
```

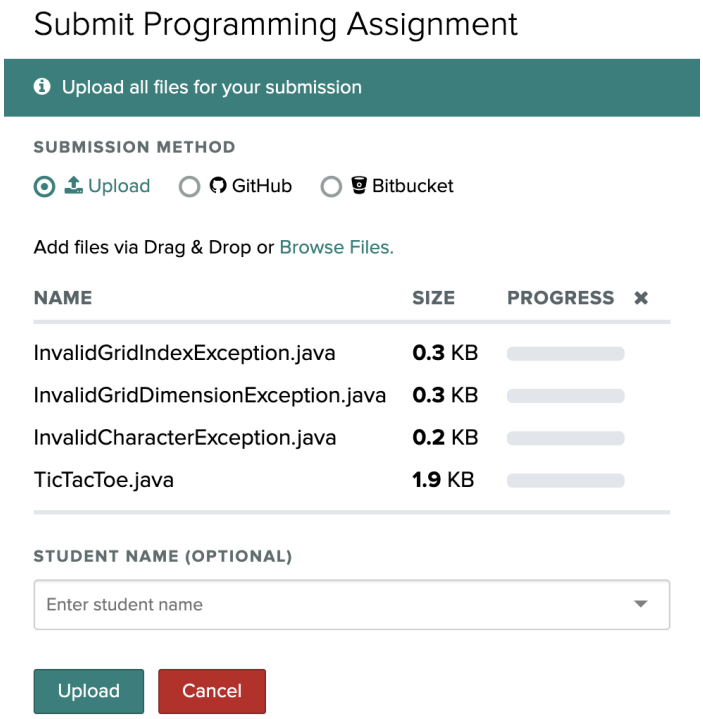
Submission

You're almost there! Please follow the instructions below carefully and use the **exact submission format**. Because we will use scripts to grade, **you may receive a zero** if you do not follow the same submission format.

1. Go to Gradescope via Canvas and click on Assignment 5.
2. Click the DRAG & DROP section and directly select the required files (`TicTacToe.java`, `InvalidCharacterException.java`, `InvalidGridIndexException.java`, and `InvalidGridDimensionException.java`). Drag & drop is fine. Please make sure you do not submit a zip, just the four files in one Gradescope submission. Make sure the names of the files are correct.
3. You can resubmit an unlimited number of times before the due date. Your score will depend on your final (most recent) submission, even if your former submissions have higher scores.
4. The autograder is for grading your uploaded files automatically. Make sure your code can compile on Gradescope.

   NOTE: The Gradescope Autograder you see is a minimal autograder. For this particular assignment, it will only show the compilation results and the results of basic tests. After the assignment deadline, a thorough Autograder will be used to determine the final grade of the assignment. **Thus, to ensure that you would receive full points from the thorough Autograder, it is your job to extensively test your code for correctness via `unitTests`.**

5. Your submission should look like the screenshot below. **If you have any questions, then feel free to post on Piazza!**


Submit Programming Assignment
Upload all files for your submission
SUBMISSION METHOD
Upload
GitHub
Bitbucket
Add files via Drag & Drop or Browse Files.
NAME
SIZE
PROGRESS
InvalidGridIndexException.java
0.3 KB
InvalidGridDimensionException.java
0.3 KB
InvalidCharacterException.java
0.2 KB
TicTacToe.java
1.9 KB
STUDENT NAME (OPTIONAL)
Enter student name
Upload
Cancel