German University in Cairo
Media Engineering and Technology
Prof. Dr. Slim Abdennadher

Computer Programming Lab, *Spring 2015*
Yu Gi Oh: **Milestone 3**

Deadline: Fri 10.4.2015 @23:59

This milestone is an *exercise* on **handling exceptions**. The *objective* of this milestone is to handle the errors and the special cases in the game. Building customized exceptions is useful because you can pass all the data you need to the object to facilitate the exception handling.

By the **end of this milestone**, you should have:

- A customized definition of the described exceptions
- An implementation for handling them to support the game logic

1 Update the CSV file reading methods

The *objective* of this part is to increase the robustness of the CSV file reading code to ensure the integrity of the data.

1.1 Defining the path of the files

1. Update `Deck` class by adding **two** new *static* instance parameters
   (a) `String monstersPath`: path to the CSV file containing the monsters.
   (b) `String spellsPath`: path to the CSV file containing the spells.
2. The **default value** for the variables should be the paths of the CSV files in your project.
3. The new instance variables are **private**. You should interact with them using getters and setters only.

1.2 Handling missing database files

1. Handle the `FileNotFoundException` whenever you cannot find the file or cannot open it. You should **handle** this exception by asking the user to enter a new path. The user has a *maximum* of 3 trials before the exception is thrown.

   Example 1 *You could not find the CSV file using the default path because the name is incorrect:*
   ```
   Please enter a correct path: hj.csv
   The file was not found
   ```

`Please enter a correct path:` *ui.csv*
`The file was not found`
`Please enter a correct path:` *xc.csv*
Throw the exception and show the stack trace

2. You should take the input from the user using **only** the `Scanner` object. The scanner instance **must** be initialized only once for *all* the three trials.

3. You should **not** handle the `IOException` resulting from the opening, reading of lines or closing of the buffered reader. However, it should be thrown.

1.3 Handling malformed data

CSV files can contain malformed data. *Malformed data* is data that does not follow the format you expect. Malformed data is problematic because you will not be able to parse it. You are required to **create** and **handle** a hierarchy of *five custom exception classes* to handle the different cases of malformed data.

General requirements

1. Your exception should be placed in `eg.edu.guc.yugioh.exceptions` package.
2. The **constructor** of the exception should initialize all the attributes of the exception.
3. All your instance variables are private. You are allowed to interact with them using getters and setters only.
4. You should **handle** the following exceptions by:
   - Updating the respective values of the instance parameters in the exception (*E.g.: line number*)
   - Ask the user to enter a new path for an uncorrupted file.
   - The program should try to read the new file again correctly.
   - **After 3 failures**, the exception should be thrown. *Please note* that the 3 failures are the total number of trials for failing to load the file regardless the exception. You should take the input from the user using **only** the `Scanner` object. The scanner instance **must** be initialized only once for *all* the three trials.

     Example 2 *Consider the following scenario for reading the monsters CSV file:*

     – ***Default try****: Your program tries to read the files using the default paths and the file is not found* → *`FileNotFoundException` is thrown and the user is granted a chance to re-enter the path.*

     – ***Try (1)****: The user enters an incorrect path for the file* → *`FileNotFoundException` is thrown and the user is granted another chance to re-enter the path.*

     – ***Try (2)****: The user enters a correct path but the first line has a missing entry* → *`MissingFieldException` is thrown and the user is granted another chance to re-enter the path.*

- ***Try (3):*** *The user enters a correct path but the third line has an empty field →* `EmptyFieldException` *is thrown.*

Exceptions hierarchy

1. Create `UnexpectedFormatException` class that represents the existence of malformed data in the CSV file.

   (a) The exception is a subclass of the `Exception` class.

   (b) It has 2 instance variables:

   - `String sourceFile`: Name of the CSV file causing the exception
   - `int sourceLine`: Line number of the malformed entry

   (c) You should have a **constructor** that takes the following inputs in the mentioned order: *source file and source line*

2. **Create the following exceptions to represent the possible types of the** `UnexpectedFormatException` **exception**

   (a) `MissingFieldException`: Missing fields should throw the exception. You should have a **constructor** that takes the following inputs in the mentioned order: *source file and source line*

   Example 3 *Malformed entry:* ***spell name,spell description***

   (b) `EmptyFieldException`: Empty fields should throw the exception (`SPACE` is also considered empty). You should have a **constructor** that takes the following inputs in the mentioned order: *source file, source line and source field*

   Extra instance variable:

   - `int sourceField`: Index of the empty field. Please note that the *index* of the first field is 1 not 0

   Example 4 *Malformed entry:* ***Spell, ,spell description***

   (c) `UnknownCardTypeException`: Incorrect card types should throw the exception. You should have a **constructor** that takes the following inputs in the mentioned order: *source file, source line and unknownType*

   Extra instance variable:

   - `String unknownType`: The type found in the CSV entry

   Example 5 *Malformed entry:* ***Trap,spell name,spell description***

   (d) `UnknownSpellCardException`: Other undefined spell types should throw the exception. You should have a **constructor** that takes the following inputs in the mentioned order: *source file, source line and unknownSpell*

   Extra instance variable:

   - `String unknownSpell`: The spell name found in the CSV entry

   Example 6 *Malformed entry:* ***Spell,CustomSpell,spell description***

2 Enforce the game regulations

You should **implement** the following list of customized exceptions. You **must** use the same names for the exceptions described in the requirements. You are allowed to add more exceptions, if needed.

This part could also be implemented using **defensive programming**, i.e. you use if statements to enforce the rules and prevent the violations. However, you are required to implement it using exceptions as an exercise on the concept.

General requirements

1. **All** the exceptions should be placed in the `eg.edu.guc.yugioh.exceptions` package.
2. **All** exceptions are subclasses of the `RuntimeException` class.
3. You are **not required** to handle any of **following** exceptions in this milestone.
4. You must **throw** them whenever needed.

Required custom exceptions

1. `NoSpaceException`: This exception is thrown whenever a player tries to add extra cards to the field that exceed the number of available slots. It has 2 subtypes:
   (a) `NoMonsterSpaceException`: The exception is fired whenever there is no space for more monsters in the field.
   (b) `NoSpellSpaceException`: The exception is fired whenever there is no space for more spells in the field.
2. `WrongPhaseException`: This exception is fired when the player tries to do the actions of a phase in another phase.
3. `MultipleMonsterAdditionException`: The exception is fired when the player tries to add more than one monster to the field. **Remember that** *you are allowed to add only one monster to the field in any mode per turn.*
4. `MonsterMultipleAttackException`: The exception is fired when the player tries to attack more than once in the same turn using the same monster.
5. `DefenseMonsterAttackException`: The exception is fired when the player tries to attack with a monster in **DEFENSE** mode.