Programming Exercise 05 – Movie Theater

Authors: Joshua

Problem Description

This assignment will assess your knowledge of exceptions in Java.

You and your friends love to watch movies at the local movie theater! You want to make sure that you will be able to see all the new movies, but you don't want to watch any you've already seen. Luckily, you know how to use Java to determine if you should go! For PE5, you will be creating your own exceptions. You will be simulating a movie theater with various movies. Problems may arise when trying to see a movie that isn't being shown or if you have already seen a movie at this theater.

Solution Description

You will create three classes, two of which are exception classes: `MovieTheater.java`, `FilmNotFoundException.java`, and `AlreadyWatchedException.java`.

Note: You must reuse methods when it is possible to do so.

FilmNotFoundException.java

This should be a **checked** exception.

Constructors:

- `FilmNotFoundException(String movie)`
  - This exception should have the message "[movie] is not playing at this movie theater."
  - **Hint:** How can we use the super class's constructor here?

AlreadyWatchedException.java

This should be an **unchecked** Exception.

Constructors:

- `AlreadyWatchedException()`
  - This exception should have the message "You've already seen this movie here!"
  - **Hint:** How can we use the super class's constructor here?

MovieTheater.java

This class represents a movie theater playing different movies.

Variables:

All variables should not be visible to outside classes.

- `ArrayList<String> movies` – the movies shown at the theater.
- `ArrayList<String> watched` – the movies you have already watched at the theater.

Constructors:

- `MovieTheater(ArrayList<String> movies, ArrayList<String> watched)`
    - Use the ArrayList copy constructor to assign copies of the arguments `movies` and `watched` to their respective instance variables. Creating a copy of the lists passed in prevents the caller from changing the state of the lists from "outside" the scope of the object.

Methods:

- `throwIfMoviesMissing(ArrayList<String> interestingMovies)`
    - This method will look to see if **ALL** of the movies you are interested in seeing are played at the theater.
    - A `FilmNotFoundException` should be thrown on the first occurrence of a movie in `interestingMovies` not being found within `movies`. The exception message should be related to the movie in question.
        - **HINT:** We are required to handle this exception at **compile-time**. Think about what this implies about the method header of this method. The exception must be allowed to propagate outside this method and should **NOT** be caught within this method.
        - **ANOTHER HINT:** The ArrayList class has a *contains()* method that will be *very* helpful here
    - If all items present in `interestingMovies` are at the movie theater, don't throw any exceptions.
    - This method will return nothing.

- `watchMovie(String movie)`
    - In this method, you will watch a movie at the theater if you haven't already watched it there.
    - To watch a movie, it should be removed from `movies` and added to `watched`.
    - If the passed in movie isn't played at the theater, throw a `FilmNotFoundException`.
        - **Hint:** We are required to handle this exception at **compile-time.** Think about what this implies about the method header of this method.
    - If you have already watched the passed in movie at this theater, throw an `AlreadyWatchedException`.
    - This method should return nothing.

- `selectRecommended(ArrayList<String> recommendedMovies)`
    - Create a new ArrayList called `willSee`.
    - Using the list of movies recommended by a friend, determine which movies you will see at the MovieTheater.
    - This ArrayList `willSee` should hold only the items present within both `recommendedMovies` and the movies being shown at the MovieTheater.
    - Return this new ArrayList at the end of the method.

- `main(String[] args)`
    - Create an ArrayList of Strings named `movies`. Initialize this list with at least 5 String elements, each representing the name of a movie playing at a local theater.

- Create another ArrayList of Strings named `watched`. Initialize this list with at least 2 movies that aren't present in `movies`.
- Create an instance of `MovieTheater`. Pass both `movies` and `watched` into the constructor.
- Inside a try/catch block,
  - Call `throwIfMoviesMissing(...)` and pass in the `movies` list you created.
  - Call `watchMovie(...)` and pass in a movie title. Try this for both one that's in the list of movies and with one that isn't in the list of movies.
  - Call `selectRecommended(...)` and pass in a list with titles.
  - Catch any `FilmNotFoundxceptions`. If one of these is caught, get the error message using getMessage() and print it to the console.
  - Add an additional catch for `AlreadyWatchedExceptions.` If one of these is caught, get the error message using getMessage() and print it to the console.
  - Don't catch **ANY** other types of Exceptions.
  - Regardless of if an exception occurs or doesn't occur, print out "`Took a look at the movies!`" to the console.
    - **Hint:** what block can we add to the try-catch to make this print happen in all situations?)
- Continue testing by adjusting the contents of the lists passed into different methods to see how your code handles different cases!

Sample Outputs

Please refer to the PE05 clarification thread on Ed Discussion for sample outputs.
HINT: To help debug your code, try inserting print statements in places where variables are changed.

Feature Restrictions

There are a few features and methods in Java that overly simplify the concepts we are trying to teach or break our autograder. For that reason, do not use any of the following in your final submission:

- `var` (the reserved keyword)
- `System.exit`

Import Restrictions

You may only import the following:

- `java.util.ArrayList;`

Collaboration

Only discussion of the PE at a conceptual high level is allowed. You can discuss course concepts and HW assignments broadly, that is, at a conceptual level to increase your understanding. If you find yourself dropping to a level where specific Java code is being discussed, that is going too far. Those discussions should be reserved for the instructor and TAs. To be clear, you should never exchange code related to an assignment with anyone other than the instructor and TAs.

Important Notes (Don't Skip)

1. Non-compiling files will receive a 0 for all associated rubric items
2. Do not submit `.class` files.

3. Test your code in addition to the basic checks on Gradescope
4. Submit every file each time you resubmit
5. Read the "Feature Restrictions" and "Import Restrictions" to avoid losing points
6. Check on the class forum for all official clarifications
7. It is expected that everyone will follow the Student-Faculty Expectations document, and the Student Code of Conduct. The professor expects a positive, respectful, and engaged academic environment inside the classroom, outside the classroom, in all electronic communications, on all file submissions, and on any document submitted throughout the duration of the course. **No inappropriate language is to be used, and any assignment, deemed by the professor, to contain inappropriate, offensive language or threats will get a zero**. You are to use professionalism in your work. Violations of this conduct policy will be turned over to the Office of Student Integrity for misconduct.

Checkstyle

Starting from PE03, you will begin implementing Checkstyle on your submission. (To learn more about Checkstyle, check out cs1331-style-guide.pdf under CheckStyle Resources in the Modules section of Canvas.) If you don't have Checkstyle yet, download it from Canvas -> Modules -> CheckStyle Resources -> checkstyle-8.28.jar. Place it in the same folder as the files you want to run Checkstyle on. Run checkstyle on your code like so:

```
$ java -jar checkstyle-8.28.jar yourFileName.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. If you had any errors, they would show up above this message, and the number at the end would be the points we would take off (limited by the Checkstyle cap mentioned above). In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

Turn-In Procedure

Submission

To submit, upload the files listed below to the corresponding assignment on Gradescope:

- `FilmNotFoundException.java`
- `AlreadyWatchedException.java`
- `MovieTheater.java`

Make sure you see the message stating "PE05 submitted successfully". From this point, Gradescope will run a basic autograder on your submission as discussed in the next section. **Any autograder tests are provided as a courtesy to help "sanity check" your work and you may not see all of the test cases used to grade your work.** Autograder tests are **NOT** guaranteed to be released when the assignment is released, so ***YOU*** are responsible for thoroughly testing your submission on your own to ensure you have fulfilled the requirements of this assignment. If you have questions about the requirements given, reach out to a TA or Professor via our class discussion forum for clarification.

You can submit as many times as you want before the deadline, so feel free to resubmit as you make substantial progress on the assignment (submit early and often). We will only grade your last submission, so be sure to **submit every file each time you resubmit**.

Gradescope Autograder

For each submission, you will be able to see the results of a few basic test cases on your code. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue.

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g. forgetting checkstyle, non-compiling code)
- Provide basic formatting and usage validation

In other words, the test cases on Gradescope are by no means comprehensive. Be sure to thoroughly test your code by considering edge cases and writing your own test files. You also should avoid using Gradescope to compile, run, or checkstyle your code; you can do that locally on your machine. (We are not using checkstyle for this programming exercise, but in future HWs we will).
Other portions of your assignment can also be graded by a TA once the submission deadline has passed, so the output on Gradescope may not necessarily reflect your grade for the assignment.