# Programming Exercise 05 – The Movie Theater

Authors: Joshua, Daniel, Nicolas, Sang Yoon
Topics: Exceptions

## Problem Description

Please make sure to read all parts of this document carefully.

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 PE05, 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 need to complete and turn in 3 classes, two of which are exception classes: MovieTheater.java, FilmNotFoundException.java, and AlreadyWatchedException.java.

**Notes:**

1. All variables should be inaccessible from other classes and require an instance to be accessed through, unless specified otherwise.
2. All methods should be accessible from everywhere and must require an instance to be accessed through, unless specified otherwise.
3. Use constructor chaining and reuse methods whenever possible! Ensure that your constructor chaining helps you maximize code reuse!
4. Reuse code when possible and helper methods are optional.
5. Make sure to add Javadoc comments to your methods and classes!

### *FilmNotFoundException.java*

This should be a **checked** Exception. Note that this class should be a child of the most generic checked exception class in the `java.lang` package.

**Constructors:**

- `FilmNotFoundException(String movie)`
  - This exception should contain 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. Note that this class should be a child of the most generic unchecked exception class in the `java.lang` package.

**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.
  - If `movies` is null, default it to an empty `ArrayList`
  - If `watched` is null, default it to an empty `ArrayList`

**Methods:**

- `throwIfMoviesMissing(ArrayList<String> interestingMovies)`
  - In this method, you will look to see if **ALL** the movies you are interested in seeing are playing at the theater.
  - This method should not return anything.
  - If `interestingMovies` is null, throw an `IllegalArgumentException` with a helpful message.
  - A `FilmNotFoundException` should be thrown on the first occurrence of a movie in `interestingMovies`, that is not found within `movies`. The exception message should be related to the movie in question.
    - **Hint #1**: 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.
    - **Hint #2**: 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.
- `watchMovie(String movie)`
  - In this method, you will watch a movie at the theater if you haven't already watched it there.
  - This method should not return anything.
  - To watch a movie, it should be removed from `movies` and added to `watched`.
  - If `movie` is null, throw an `IllegalArgumentException` with a helpful message.
  - 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`.
  - Make sure to throw the `FilmNotFoundException` before the `AlreadyWatchedException`.
- `selectRecommended(ArrayList<String> recommendedMoives)`
  - In this method, using the list of movies recommended by a friend, you will determine which movies you would see at the MovieTheater.
  - If `recommendedMovies` is null, throw an `IllegalArgumentException` with a helpful message.
  - Create a new `ArrayList` called `willSee`.
  - This `ArrayList willSee` should hold only the items present within both `recommendedMovies` and the movies being shown at the MovieTheater that have not yet been watched.
  - 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 a second `ArrayList` of `Strings` named `watched`. Initialize this list with at least 1 movie present in `movies` and at least 1 movie that isn't present in `movies`.
  - Create a third `ArrayList` of `Strings` named `recommended`. Initialize this list with at least 1 movie present only in `movies`, at least 1 movie present only in `watched`, and at least 1 movie in neither of the previous `ArrayLists`.
  - 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. Call this method with a movie title that is in the list of `movies`, one that is exclusively in `watched`, and one that is in neither.
    - Call `selectRecommended()` and pass in the `recommended` list you created.
      - Iterate through the resulting `ArrayList` and print out each element on a new line.
    - Catch any `FilmNotFoundExceptions`. 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!

## Checkstyle

You must run Checkstyle on your submission (to learn more about Checkstyle, check out cs1331-style-guide.pdf under the Checkstyle Resources module on Canvas). **The Checkstyle cap for this assignment is 35 points.** This means there is a maximum point deduction of 35. 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 number of points we would take off (limited by the Checkstyle cap). In future assignments we will be increasing this cap, so get into the habit of fixing these style errors early!

Additionally, you must Javadoc your code.

Run the following to only check your Javadocs:

```
$ java -jar checkstyle-8.28.jar -j yourFileName.java
```

Run the following to check both Javadocs and Checkstyle:

```
$ java -jar checkstyle-8.28.jar -a yourFileName.java
```

For additional help with Checkstyle see the CS 1331 Style Guide.

## Turn-In Procedure

### *Submission*

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

- `FlimNotFoundException.java`
- `AlreadWatchedException.java`
- `MovieTheater.java`

Make sure you see the message stating the assignment was 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 the test cases used to grade your work.** 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 the class 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. We will only grade your latest submission. **Be sure to submit every file each time you resubmit**.

### *Gradescope Autograder*

If an autograder is enabled for this assignment, you may be able to see the results of a few basic test cases on your code. Typically, tests will correspond to a rubric item, and the score returned represents

the performance of your code on those rubric items only. If you fail a test, you can look at the output to determine what went wrong and resubmit once you have fixed the issue. **We reserve the right to hide any or all test cases, so you should make sure to test your code thoroughly against the assignment's requirements.**

The Gradescope tests serve two main purposes:

- Prevent upload mistakes (e.g., 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.

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.

## *Burden of Testing*

You are responsible for thoroughly testing your submission against the written requirements to ensure you have fulfilled the requirements of this assignment.

Be **very careful** to note the way in which text output is formatted and spelled. Minor discrepancies could result in failed autograder cases.

If you have questions about the requirements given, reach out to a TA or Professor via the class forum for clarification.

## Allowed Imports

- `java.util.ArrayList`

## 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`
- `System.arraycopy`

## Collaboration

Only discussion of the assignment at a conceptual high level is allowed. You can discuss course concepts and 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)

- Non-compiling files will receive a 0 for all associated rubric items.

- Do not submit `.class` files.
- Test your code in addition to the basic checks on Gradescope.
- Submit every file each time you resubmit.
- Read the "Allowed Imports" and "Restricted Features" to avoid losing points.
- **Check on Ed Discussion for a note containing all official clarifications and sample outputs.**

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.