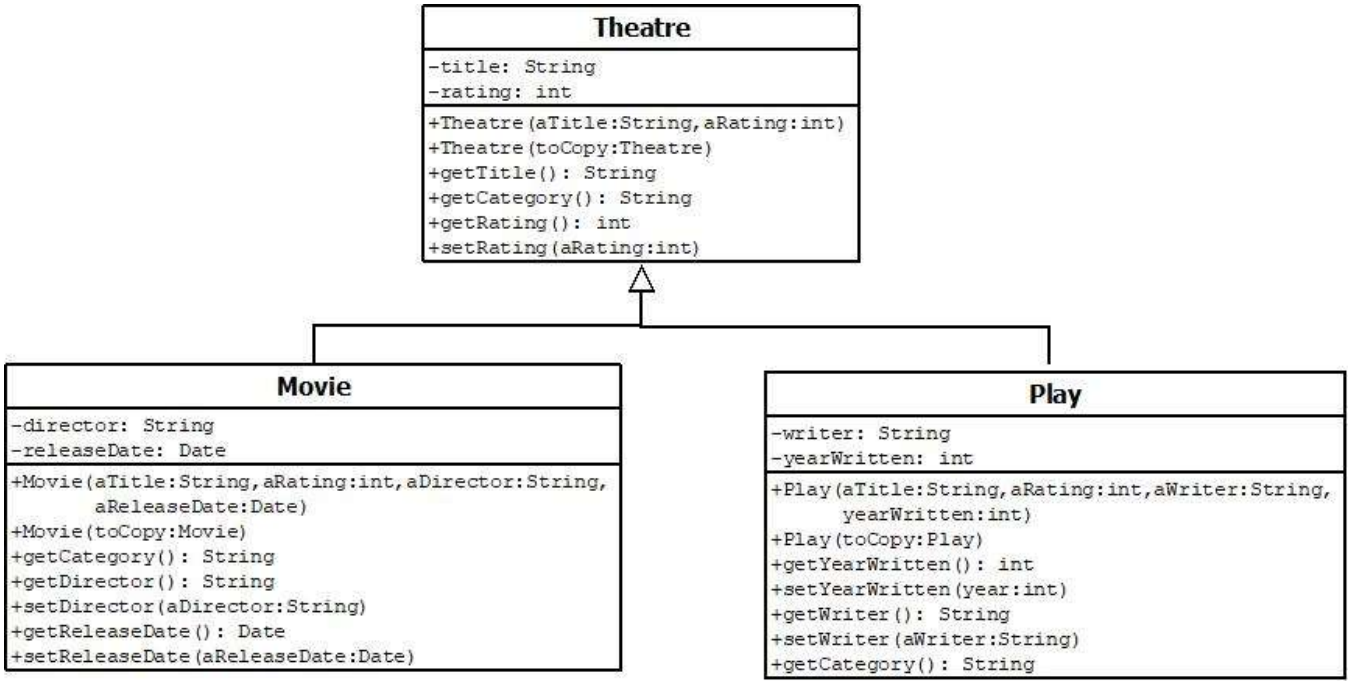
CPSC 219 – Coding Challenge 3 – Practice 1

This is a practice coding challenge. You may ask other students and your TA any questions you wish. Note that you are expected to complete the actual coding challenge independently. Having a solution for this coding challenge will not help you come up with your own solution for the actual coding challenge. Only if you can solve this practice independently can you be confident that you can complete the actual coding challenge successfully. See previous coding challenges for rules for coding challenges and best practices for success.

Remember to always create a 'skeleton' first and compile and test this using the provided JUnit test. For this coding challenge, you will be asked to create two or three classes. Make sure you create a skeleton of all classes first and place them in a zip file for submissions to WebCAT.

Requirements for Skeleton of Theatre, Movie and Play



Additional Requirements

Theatre

***title*:** The instance should always store the string in all upper case, if the method is called with a string that contains any lower case characters, the lower case characters should be converted to upper case.

rating must be a value between 0 and 10 (inclusive). If the rating provided is outside of that range, the rating should remain unchanged.

getCategory: A movie is considered A category if its rating is 9 or 10, B category if the rating is 7 or 8, C category if the rating is 5 or 6, D category if the rating is 3 or 4 and F category otherwise.

Movie

releaseDate*:* Is at most one year in the future. Otherwise leave unchanged. If no date set at all set the release date to the current date.
getCategory*:* Override from parent to precede with release date and '-'. Call parent getCategory to accomplish this, do not call getRating from parent.

Play

yearWritten*:* must be before 2019.
getCategory*:* Override from parent to return 'Classic' if it was written more than 200 years ago, 'Contemporary' if it was written more than 50 years ago and 'Modern' otherwise.

Using the Date class

- Import from java.util
- The only Date class you need to create is one that represents now, which is returned by the default constructor in the Date class. All other dates will be passed as arguments. (Note that this may be null).
- The method getTime() in the Date class returns the number of milliseconds since the Epoch. If you have two Date objects d1 and d2, with d1 a later date, the number of milliseconds between the two is d1.getTime() – d2.getTime().
- Use boolean methods before(Date) and after(Date) to compare two dates. For example, d1.before(d2) will return true if d1 appears in time before d2.

Notes: Do not duplicate instance variables from parent class in child class. Instead invoke appropriate super constructor and methods in parent class.