EECS2030F: LAB 5 Due: Nov. 11, 2025 - 11:59 pm

The purpose of this lab is to ensure that you practice

A. Using inheritance and polymorphism with method overriding.
B. Understanding static and dynamic binding.
C. Implementing custom exception classes by extending Exception superclass.
D. Working with file I/O and parsing structured data using Scanner.
E. Using polymorphic collections (ArrayList<ParentType>).

1. Lab Policies

a) **Academic Integrity**: Submit your own work. Do not copy code from classmates or online sources. All violations will be reported as academic misconduct.

b) **Submission Format**: Submit only the file: **MediaLibrary.java**. Do not upload ZIP files or project folders.

c) **Work Environment**: Complete the methods and test your submission by using tester file**.** Your code should not contain a main method and Scanner Class objects and its methods. Ensure your code compiles without errors.

d) **Deadline:** Submit your **MediaLibrary.java** file to the eClass course page by **Tuesday, November 11(11:59pm).** No late submissions are accepted. Email submissions are not accepted.

e) Don't change the name of the file as we will not be able to test your code with our tester**.** It needs to be **MediaLibrary.java** (no case change)

f) Your lab assignment is not graded during the weekly lab sessions scheduled. The lab sessions are meant to get your questions answered from TAs.

2. Downloading and importing the Starter Project

You have already done this step in previous labs. Follow the below listed steps to set up your working environment by:

a) Download the Eclipse Java project archive file from eClass: EECS2030_Lab5.zip

b) Launch Eclipse and browse to EECS2030-workspace (for instance or your own created workspace).

c) In Eclipse:

- Choose File->Import

- Under General, choose Existing Projects into workspace
- Choose Select archive file. Browse your compressed zip folder and attach it.
- Make sure that the EECS2030_Lab5 box is checked under Projects and you don't have the same project already in the workspace. Then Finish.
- You should see two files, one is called **MediaLibrary.java** and one **MediaLibraryTest.java**.

Note: You will see errors in both the files as the methods are not implemented. The moment you start implementing the code correctly, the errors will go away.

3. Important Notes:

To practice testing, we have only provided a set of incomplete test cases. Please make sure to add enough test cases to the tester that thoroughly tests your code. Look at the tester code, **MediaLibraryTest.java**, and add more test cases to ensure comprehensive testing of your code. To do this, you can copy one of the methods, change the name of the method to avoid having a duplicate method name, and modify the body of the method to test your code with your selected input.

4. Javadoc generation

The Javadoc has been written for you. All you need to do is generate it as an HTML file to make navigation easier. To do this, click on the lab5 package, select Project -> Generate Javadoc. It will ask you for the location where you want to store the documentation. You can choose the default location or select your own folder. Enter the path, and then click on Finish. If you look at the location where you stored the documentation, you'll see a file called index.html. Clicking on this file will display the project documentation in your browser.

If you have any doubts on the Javadoc generation, please review lecture slides of week2.

5. Programming Tasks for this Lab

In this lab, you are going to implement a **Digital Media Library Management System** that demonstrates inheritance, polymorphism, method overriding, dynamic binding, and exception handling. You will create a parent class (Media) representing general media properties, three

subclasses for specialized media types (Book, Movie, Music), a library management class (MediaLibrary), and two custom exception classes.

Understanding the Class Hierarchy

The system is built around a central Media parent class that contains common attributes shared by all types of media items. This parent class defines fundamental properties such as title, creator, and year of release that every media item needs regardless of its specific type. From this parent class, three specialized subclasses extend the basic media functionality using inheritance. The Book subclass represents books, adding properties specific to books such as ISBN and number of pages. The Movie subclass represents movies, adding properties for genre and duration. The Music subclass represents music albums, also adding genre and duration properties.

Understanding Polymorphism and Dynamic Binding

Polymorphism allows you to treat objects of different subclasses uniformly through references to their parent class. In this lab, the MediaLibrary class uses an ArrayList<Media> to store different types of media objects (books, movies, and music). Even though the ArrayList is declared to hold Media references, it can actually store Book, Movie, and Music objects because they all inherit from Media.

Dynamic binding (also called late binding) means that when you call a method on a Media reference, Java determines at runtime which version of the method to execute based on the actual object type. For example, when you call getMediaType() on a Media reference that points to a Book object, Java will execute Book's version of getMediaType(), not Media's version. This happens automatically at runtime.

Understanding Custom Exceptions

Exceptions are objects that represent error conditions. Java provides many built-in exceptions, but you can create your own by extending the Exception class. In this lab, you will create two custom exceptions: InvalidMediaFormatException (thrown when file data format is incorrect) and MediaNotFoundException (thrown when a requested media item cannot be found). By extending the Exception class, your custom exceptions inherit all the functionality of exceptions while adding specific behaviors for your application.

Implementation Requirements

For this lab, you will implement the following methods from each class in a SINGLE file named MediaLibrary.java. Do check the tester file to understand the format of return types for the methods.

Task 1: Implementing Custom Exception Classes

Implement two custom exception classes by extending the Exception class.

A) InvalidMediaFormatException: This exception should be thrown when the data format in the file is incorrect.
B) MediaNotFoundException: This exception should be thrown when a requested media item cannot be found in the library.

Task 2: Implementing Media Parent Class

Implement the Media class as the parent class representing common properties of all media types.

Methods to implement:

- String getDisplayInfo() - returns formatted string: "Media: [title] by [creator] ([year])"
- double getPopularityScore() - returns basic score: (2025 - yearReleased) * 1.0
- boolean equals(Object obj) - compare media items by title and creator (case-insensitive)

Important: Subclasses will override getMediaType(), getDisplayInfo(), and getPopularityScore() to provide specific implementations. **This demonstrates polymorphism!**

Task 3: Implementing Book Class

Implement the Book class that extends Media and adds book-specific properties.

Methods to implement:

- Override getDisplayInfo() - returns: "Book: [title] by [author] ([year]) - ISBN: [isbn], Pages: [pages]"
- Override getPopularityScore() - calculate as: (2025 - yearReleased) * 0.5 + (numberOfPages / 100.0)

Important: When you override these methods, Java uses dynamic binding to call the correct version at runtime!

Task 4: Implementing Movie Class

Implement the Movie class that extends Media and adds movie-specific properties.

Methods to implement:

- Override getDisplayInfo() - returns: "Movie: [title] directed by [director] ([year]) - [duration] mins, Genre: [genre]"

- Override getPopularityScore() - calculate as: (2025 - yearReleased) * 0.8 + (duration / 10.0)

Task 5: Implementing Music Class

Implement the Music class that extends Media and adds music-specific properties.

Methods to implement:

- Override getDisplayInfo() - returns: "Music: [title] by [artist] ([year]) - [duration] mins, Genre: [genre]"
- Override getPopularityScore() - calculate as: (2025 - yearReleased) * 0.6 + (duration / 5.0)

Note: Book, Movie, and Music all override the same methods from Media but provide different implementations. **This is polymorphism!**

Task 6: Implementing MediaLibrary Class

Implement the MediaLibrary class that manages a collection of media items using polymorphism!

Methods to implement:

A) void loadMediaFromFile(String filename) throws InvalidMediaFormatException, FileNotFoundException

- Reads media data from the specified file using Scanner
- Parses each line using String.split("\\|") to handle the pipe-delimited format
- Creates appropriate Media objects (Book, Movie, or Music) based on the first field
- Adds them to mediaCollection
- Throws InvalidMediaFormatException if format is incorrect (wrong number of fields, invalid media type, or NumberFormatException when parsing integers)
- Throws FileNotFoundException if file doesn't exist
- Uses try-catch-finally to ensure Scanner is closed
- Example: Line "BOOK|1984|George Orwell|1949|978-0451524935|328" creates Book object

Important: Even though mediaCollection is ArrayList<Media>, you can add Book, Movie, and Music objects because of inheritance!

B) void addMedia(Media media): Adds a media item to the collection. Throws IllegalArgumentException if media is null.

C) Media findMediaByTitle(String title) throws MediaNotFoundException

- Searches for media with exact title (case-insensitive)
- Returns the Media object if found
- Throws MediaNotFoundException if not found (include the searched title)

D) ArrayList<Media> getMediaByType(String mediaType)

- Returns ArrayList of all media items of specified type ("Book", "Movie", or "Music")
- Use getMediaType() method to filter - demonstrates DYNAMIC BINDING!
- When you call media.getMediaType(), Java determines at runtime whether to call Book's, Movie's, or Music's version

E) Media getMostPopularMedia()

- Returns the Media item with highest popularity score
- Uses polymorphic call to getPopularityScore() - dynamic binding
- Each media type calculates popularity differently
- Returns null if collection is empty

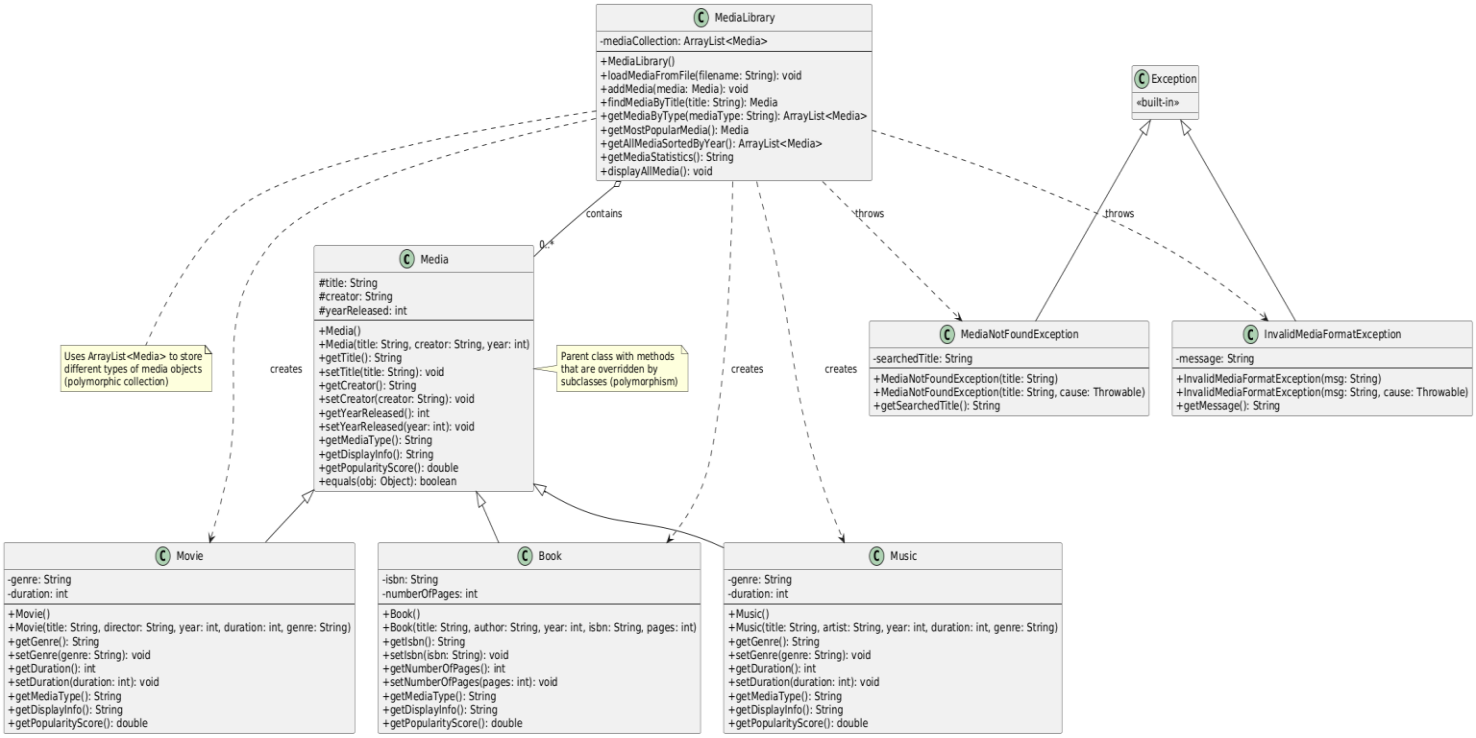
F) ArrayList<Media> getAllMediaSortedByYear()

Returns ArrayList of all media items sorted by year (oldest first). Don't modify the original collection.

G) String getMediaStatistics()

- Returns a String with statistics:
- - Total number of items
- - Number of books
- - Number of movies
- - Number of music albums
- - Average year of release
- Use getMediaType() to count each type - demonstrates polymorphism!

UML Diagram:



6. Testing your code

A test case file has been provided to evaluate your methods. Your implementation should pass all the test cases if it correctly handles both normal and edge cases. **Do not modify or update the provided test cases in any way. You should create your own test cases too to thoroughly test your code.**

A sample data file media_data.txt has been provided in the project. You can use this file to test your loadMediaFromFile() method manually**.**

7. Submit

Submit only one file named " **MediaLibrary.java** " via eClass by clicking on the lab link.

8. Marking Schema

You are graded based on the correctness of your code. For each method, there will be comprehensive test cases to examine the correctness of the code. All test cases carry the same weight.

Please note that in all the labs, even if the test cases are provided, we will test your code with a different set of test cases to ensure that you have tested your code completely.

Note:

- The program that does not compile will get zero marks.
- Don't change the method headers. The test cases will fail if you change the method headers.
- Your submission should strictly have the name as **MediaLibrary.java**, otherwise it can not be used to test your assignment. It will have zero grades.
- No resubmissions are allowed. Therefore, please make sure you submit the correct file within due date.

****