**Howard University**

**College of Engineering and Architecture**

**Department of Electrical Engineering & Computer Science**

**Large Scale / Object-Oriented Programming**

**Spring 2025**

**Midterm Exam**

**100 pts.**

March 20, 2025

**Instructions:**

1. **OPEN BOOK, OPEN NOTES. THERE IS NO COLLABORATION ON THIS EXAM. Failure to adhere to this standard will result in failure for this class and possible academic charges.**
2. **Submit completed exam to your github repository. Verify that the commit(s) completed successfully**
3. **All Java classes should be well-commented (javadocs) and follows best practices. You do not have to generate the html file. Failure to comply will result in deductions.**
4. **Site ALL references for anything obtained off the internet. Failure to do so will result in a zero for the question.**
5. **Commit often to your repos.**
6. **None of these question require a lot of code. Please use proper time management and don’t get caught up on one question.**

**Please upload to org.howard.edu.lsp.midterm.question1**

**(30 pts. total)**

**Given:**

**Identify CRC cards for the following set of requirements:**

The library maintains a collection of books, each having a title, author, ISBN, and availability status. Members of the library can borrow books if they are available. Each member can borrow only up to 3 books at a time. When a member returns a book, it should become available again. Librarians can add new books to the collection and remove books that are outdated or lost. The system must allow the librarian to view the list of available books and see which books are currently checked out and who has borrowed them.

Using the **Class-Responsibility-Collaborator (CRC) method**, complete the following steps:

#### ****Part 1: Identify Classes****

(10 pts.)

* Identify at least **four key classes** necessary to implement this system.
* Write a short explanation of why each class is needed.
  1. Book: The class is used to represent a book, where each individual book has title, author, ISBN and availability status attributes. It is needed to store book information for the library.
  2. Member. This class is used to represent a member of the library, allowing them to borrow and return books (only up to 3 books). It allows us to keep track of books borrowed by specific members
  3. Librarian: This class allows librarians of the library to add and remove books, while also letting them view the borrowing status of a book
  4. Library: This class holds all the book classes of the library, managing the collection and coordinating interactions between book, member and librarian classes.

#### ****Part 2: Define Responsibilities****

(10 pts.)

* For each class, list at least **two primary responsibilities** that class must have.
  1. Book:
     + Store and update its own data (title, author, ISBN, availability).
     + Provide information about its current state (e.g., whether it is available for borrowing).
  2. Member:
     + Allows borrowing books (ensuring the member does not exceed the limit)
     + Allows returning borrowed books and updates its availability
  3. Librarian:
     + Facilitates adding new books to the library collection and removing lost or outdated books
     + Views and manages the inventory by checking which books are available or borrowed (and which member currently holds the book)
  4. Library:
     + Maintains the collection of Book objects, where a Librarian can update said collection
     + Updates and tracks the status of books when members borrow or return them

#### ****Part 3: Identify Collaborators****

(10 pts.)

* For each class, identify **other classes** it interacts with to fulfill its responsibilities and explain why the collaboration is needed.
  + 1. Book Class
       - Interacts with Member class, updating Book availability to be unavailable when checked out and available when book is returned
       - Interacts with Library class as all Book objects will be held here. Also monitors book availability
    2. Member Class
       - Interacts with Book class, checking and updating Book availability when a Book is borrowed or returned
       - Interacts with Library class, allowing members to view available books while also enforcing the 3 book borrowing limit
    3. Librarian Class
       - Interacts with Library class, allowing them to update the books collection by removing or adding Book objects while also letting them check a book’s availability (in the case a book gets lost or outdated)
       - Interacts with Book class directly allowing them to add or remove books from Library collection, while also letting them update it’s availability
    4. Library Class
       - Interacts with Book class by storing and maintaining the collection of Book objects while also tracking their availability
       - Interacts with Member class by ensuring they do not exceed their borrowing limit while also displaying the books available
       - Interacts with the Librarian class, providing functionality to allow the Librarian to manage the Library collection of Book objects

Upload your response as a document (preferably embedded into this Word doc)

**Please upload to org.howard.edu.lsp.midterm.question2**

**(18 pts. total)**

**Given:**

You are given a class Book that represents a book in a library. The Book class has the following attributes:

* title (String): The title of the book.
* author (String): The author of the book.
* ISBN (String): The ISBN number of the book (a unique identifier).
* yearPublished (int): The year the book was published.

#### Task:

1. Write the Book class with appropriate constructors, getters, and setters. Be mindful of encapsulation in your declarations. (5 pts.)
2. Override the boolean equals(Object) method in the Book class so that two Book objects are considered equal if they have the same ISBN and author. (5 pts.)
3. Override the String toString() method in the Book class to return a meaningful string representation of a Book object (see sample output below) (5 pts.)
4. Create a BookDriver class that instantiates your Book class and executes the following: (3 pts.)

...

Book book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", "123456789", 1925);

Book book2 = new Book("The Great Gatsby", "F. Scott Fitzgerald", "123456789", 1925);

Book book3 = new Book("A Promised Land", "Barack Obama", "9780593239681", 2020);

System.out.println(book1.equals(book2)); **// true**

System.out.println(book1.equals(book3)); **// false**

System.out.println(book1); **// Output: Title: The Great Gatsby, Author: F. Scott Fitzgerald, ISBN: 123456789, Year Published: 1925**

…

**Please upload to org.howard.edu.lsp.midterm.question3**

**(15 pts. total)**

**Given:**

You have been asked to implement a simple **electronic voting machine** for an election. The voting machine should allow voters to cast votes for candidates and display the election results.

#### ****Requirements:****

1. The voting machine should support any number of candidates.
2. Each vote is recorded for a candidate by their name.
3. The system should prevent votes for non-existent candidates.
4. The system should be able to display the **total votes per candidate** and determine the **winner**.

You will write a class named VotingMachine and a driver named VotingMachineTest (5pts.) to test your implementation. See below.

**…**

VotingMachine vm = new VotingMachine();

// Adding candidates

vm.addCandidate("Alice"); (2.5 pts.)

vm.addCandidate("Bob");

vm.addCandidate("Charlie");

// Casting votes

vm.castVote("Alice"); (5 pts.)

vm.castVote("Alice");

vm.castVote("Bob");

vm.castVote("Charlie");

vm.castVote("Charlie");

vm.castVote("Charlie");

// Attempt to vote for a non-existent candidate

boolean success = vm.castVote("Eve"); (2.5 pts.)

System.out.println("Vote for Eve successful? " + success);

// Displaying results

System.out.println("Winner: " + vm.getWinner()); (5 pts.)

**Output: Charlie WINS with 3 votes!!**

…

**Please upload to org.howard.edu.lsp.midterm.question4**

**(25 pts. total)**

**Given:**

Create a WordProcessor class that processes a sentence and returns all longest words.

Implement a class with the following structure:

public class WordProcessor {

private String sentence; // Stores the sentence to be processed

// Constructor

public WordProcessor(String sentence) { ... }

// Method to find all longest words in the sentence

public List<String> findLongestWords() { ... }

}

#### ****Requirements:****

* The method should return **all** words with the maximum length in the **order** they appear in the sentence. (10 pts.)
* The method should handle varying amounts of whitespace between words. 5 pts.)
* You can assume that the input does not have any punctuation (e.g., periods, commas, exclamation marks).
* If the input is empty or contains no words, return an **empty list** ([]). (5 pts.)
* Write a driver, WordProcessorDriver that implements the following … (5 pts.)

…

**// Test case 1: Sentence with single longest word**

WordProcessor wp4 = new WordProcessor("This is a test with amazing results, amazing achievements!");

List<String> longestWords4 = wp4.findLongestWords();

System.out.println("Test 4 - Longest words: " + longestWords4);

// Expected Output: ["achievements"]

**// Test case 2: Sentence with multiple occurrences of the same longest word and length**

WordProcessor wp2 = new WordProcessor("Java is a powerful tool!");

List<String> longestWords2 = wp2.findLongestWords();

System.out.println("Test 2 - Longest words: " + longestWords2);

// Expected Output: ["powerful" "powerful"]

// Test case 5: Sentence with single character words

WordProcessor wp5 = new WordProcessor("A B C D E");

List<String> longestWords5 = wp5.findLongestWords();

System.out.println("Test 5 - Longest words: " + longestWords5);

// Expected Output: ["A", "B", "C", "D", "E"]

// Test case 6: Empty string input

WordProcessor wp6 = new WordProcessor("");

List<String> longestWords6 = wp6.findLongestWords();

System.out.println("Test 6 - Longest words: " + longestWords6);

// Expected Output: []

…

**Please upload to org.howard.edu.lsp.midterm.question5**

**(12 pts.)**

**Given:**

You are designing a **media streaming service** that allows users to stream different types of content, such as **music, movies, and audiobooks**. Each type of content must support the following actions:

1. **Play the content** using play(). (2 pts.)
2. **Pause the content** using pause(). (2 pts.)
3. **Stop the content** using stop(). (2 pts.)

However, different types of media have unique behaviors:

* **Movies** allow rewinding with rewind(int seconds). (1 pt.)
* **Audiobooks** support changing the playback speed with setPlaybackSpeed(double speed). (1 pt.)
* **Music** allows adding songs to a playlist with addToPlaylist(String playlistName). (1 pt.)

You should design your system **with a focus on shared behavior rather than shared state. (3 pts.)**

### ****Task:****

1. **Implement your chosen design** in Java:
   * Define a way to represent **streamable content** that ensures shared behaviors. **(2.5 pts.)**
   * Create concrete implementations for Music, Movie, and Audiobook.
   * Use additional structures if needed to support their unique behaviors.
2. **Provide a StreamingServiceDriver class** to demonstrate creating and using different media types. **(5 pts.)**

**Example Usage in StreamingServiceDriver Class:**

…

**// Create instances of different media types**

Streamable music = new Music("Still a Friend - Incognito");

Streamable movie = new Movie("Avengers End Game");

Streamable audiobook = new Audiobook("A Promised Land – Barack Obama");

// Test the common behaviors for all media types (play, pause, stop)

System.out.println("Testing common behaviors for all media types:\n");

**// Test Music**

System.out.println("Testing Music:");

music.play(); // Output: "Playing music: Still a Friend - Incognito "

music.pause(); // Output: "Paused music: Still a Friend - Incognito "

music.stop(); // Output: "Stopped music: Still a Friend - Incognito "

System.out.println();

**// Test Movie**

System.out.println("Testing Movie:");

movie.play(); // Output: "Playing movie: Avengers End Game "

movie.pause(); // Output: "Paused movie: Avengers End Game "

movie.stop(); // Output: "Stopped movie: Avengers End Game "

System.out.println();

**// Test Audiobook**

System.out.println("Testing Audiobook:");

audiobook.play(); // Output: "Playing audiobook: A Promised Land – Barack Obama"

audiobook.pause(); // Output: "Paused audiobook: A Promised Land – Barack Obama "

audiobook.stop(); // Output: "Stopped audiobook: A Promised Land – Barack Obama "

System.out.println();

**// Test unique behaviors for each media type**

**// Test Movie-specific behavior (rewind)**

Movie movieWithRewind = (Movie) movie; // Cast to Movie for specific method

movieWithRewind.rewind(30); // Output: "Rewinding movie: Avengers End Game by 30 minutes"

System.out.println();

**// Test Audiobook-specific behavior (setPlaybackSpeed)**

Audiobook audiobookWithSpeed = (Audiobook) audiobook; // Cast to Audiobook for specific method

audiobookWithSpeed.setPlaybackSpeed(1.5); // Output: "Setting playback speed of audiobook: A Promised Land – Barack Obama 1.5x"

System.out.println();

**// Test Music-specific behavior (addToPlaylist)**

Music musicWithPlaylist = (Music) music; // Cast to Music for specific method

musicWithPlaylist.addToPlaylist("Favorites"); // Output: "Added Still a Friend - Incognito to Favorites playlist"

System.out.println();

**…**