**a) String Immutability in Java**

String immutability in Java means that once a String object is created, its value cannot be changed. Any operation that appears to modify a string actually creates a new string object.

In NLU's system, this aids in:

1. **Security**: Sensitive book metadata like titles and author names can't be accidentally or maliciously altered after creation
2. **Thread-safety**: Immutable strings can be safely shared across multiple threads handling patron queries without synchronization
3. **Performance**: Frequently queried strings can be cached and reused (string interning) since they won't change
4. **Hashcode caching**: Strings cache their hashcodes, making them efficient for use in hash-based collections like HashMaps for bibliographic indexing.

**b) creating empty string objects**

// Method 1: Using string literal

String bookTitle = "";

// Method 2: Using String constructor

String authorName = new String();

// Method 3: Using empty initialization (same as method 1)

String isbnNumber = "";

// Example usage in book record initialization

public class BookRecord {

private String title = ""; // Empty string literal

private String author = new String(); // Empty string constructor

// Constructor initializing with empty strings

public BookRecord() {

this.title = "";

this.author = new String();

}

}

## c) equalsIgnoreCase() and toLowerCase() Methods

These methods improve user search functionality by:

1. **equalsIgnoreCase()**:
   * Compares strings while ignoring case differences
   * Allows patrons to find books whether they type "HARRY POTTER", "harry potter", or "Harry Potter"
   * More user-friendly as it accepts queries in any case combination
2. **toLowerCase()**:
   * Converts both search query and catalog entries to lowercase for consistent comparison
   * Can be used with equals() to achieve case-insensitive matching
   * Example: if(bookTitle.toLowerCase().equals(query.toLowerCase()))

These methods ensure patrons find books regardless of their typing case preferences, improving search success rates.

**d) Explanation of the code**

* == compares object references, not content
* author1 and author3 are different objects (one in pool, one in heap)
* equalsIgnoreCase() compares content while ignoring case differences
* String literals are interned (shared in string pool), while new String() creates new objects

**e) Book Description word counter**

import java.util.Scanner;

public class UgandaWordCounterJava {

public class UgandaWordCounter {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Prompt librarian for book description

System.out.println("Please enter the book description:");

String description = scanner.nextLine();

// Count occurrences of "Uganda" (case-insensitive)

String lowerDescription = description.toLowerCase();

String searchWord = "uganda";

int count = 0;

int index = 0;

while ((index = lowerDescription.indexOf(searchWord, index)) != -1) {

count++;

index += searchWord.length();

}

// Alternative solution using split (less efficient but simpler)

// String[] words = lowerDescription.split(searchWord);

// count = words.length - 1;

System.out.println("The word 'Uganda' appears " + count + " times in the description.");

scanner.close();

}

}

public static void main(String[] args) {

System.out.println("Hello World!");

}

}