**Practical Exercise in Java Collection Framework**

1. A university library needs a system to manage book records. Each book should have a Book ID, Title, Author, and Price.
2. The system should allow the following functionalities:
	1. Add Books to the collection.
	2. Remove a Book based on its ID.
	3. Search for a Book by title.
	4. Sort Books by Price (Lowest to Highest) using the Comparable Interface.
	5. Maintain a Stack for Tracking Borrowed Books (Last Borrowed First).
3. Develop a Java-based Library Book Management System using Generic Classes and Java Collections Framework.
4. The system should manage a collection of books, allow users to add, remove, search, and sort books based on different criteria.
5. It should also use a Stack to maintain a history of borrowed books and implement the Comparable Interface for sorting books.
6. Use Generic Classes (for book data storage), Use Collections Framework (List, Stack), and Use Comparable Interface (for sorting books).

**Basic Code Template**

import java.util.\*;

// Generic Class for Library

class Library<T> {

 private List<T> books = new ArrayList<>();

 public void addBook(T book) {

 books.add(book);

 }

 public void removeBook(T book) {

 books.remove(book);

 }

 public List<T> getBooks() {

 return books;

 }

}

// Book Class implementing Comparable

class Book implements Comparable<Book> {

 private int bookID;

 private String title;

 private String author;

 private double price;

 public Book(int bookID, String title, String author, double price) {

 this.bookID = bookID;

 this.title = title;

 this.author = author;

 this.price = price;

 }

 public String getTitle() {

 return title;

 }

 public double getPrice() {

 return price;

 }

 @Override

 public int compareTo(Book other) {

 return Double.compare(this.price, other.price);

 }

 @Override

 public String toString() {

 return "[ID: " + bookID + ", Title: " + title + ", Author: " + author + ", Price: " + price + "]";

 }

}

public class LibraryManagement {

 public static void main(String[] args) {

 Library<Book> library = new Library<>();

 Stack<Book> borrowedBooks = new Stack<>();

 // Adding books to library

 library.addBook(new Book(1, "Java Programming", "James Gosling", 599.99));

 library.addBook(new Book(2, "Data Structures", "Robert Lafore", 499.50));

 library.addBook(new Book(3, "Machine Learning", "Tom Mitchell", 799.75));

 // Display all books

 System.out.println("Books in Library:");

 for (Book book : library.getBooks()) {

 System.out.println(book);

 }

 // Sort books by price

 List<Book> sortedBooks = new ArrayList<>(library.getBooks());

 Collections.sort(sortedBooks);

 System.out.println("\nBooks Sorted by Price:");

 for (Book book : sortedBooks) {

 System.out.println(book);

 }

 // Borrow a book (Push to Stack)

 Book borrowed = sortedBooks.get(0);

 borrowedBooks.push(borrowed);

 System.out.println("\nBorrowed Book: " + borrowed);

 // Return a book (Pop from Stack)

 Book returnedBook = borrowedBooks.pop();

 System.out.println("\nReturned Book: " + returnedBook);

 }

}

**Highlights**

* Generic Class (Library<T>) – Manages books using generics.
* Comparable Interface – Sorts books by price.
* Collections Framework (List, ArrayList, Stack) – Stores and processes books.
* Stack – Manages borrowed books (LIFO order).
* Sorting with Collections.sort() – Orders books by price.

**SPOT**



### **Book Borrowing System**

* + Keep track of borrowed books in a separate collection (e.g., a HashMap<String, List<Book>> where key = student name, value = list of borrowed books).
	+ Implement a function to **return books** and update the inventory accordingly.

### **Categorization of Books using Map**

* Use a HashMap<String, List<Book>> where the key is a category/genre (e.g., Fiction, Science, History) and the value is a list of books in that category.
* Allow users to view books by category**.**