CERTIFICATE

This is to certify that [Your Name], a student of Class XII-[Section] at [Your School Name], has successfully completed the project titled **“Bank Management System”** under the supervision of [Project Supervisor Name] during the academic year [2025-26].

This project has been completed in fulfillment of the requirements for the Computer Science course and adheres to the guidelines issued by CBSE, New Delhi.

The project work meets the expected standards and has been evaluated by the undersigned.

Signature of Principal: ___________________________

Signature of Supervisor: __________________________

Signature of External Examiner: _________________

Date: ______________

Place: ______________

Student Details:

Student Names:

Class : XII

Subject: Information Technology

School Name: Torch Bearers Convent School

Affiliation By.: CBSE Board

ABSTRACT

The Library Management System (Mini) project aims to develop a computerized application to efficiently manage the daily operations of a library using basic Object-Oriented Programming concepts. This system utilizes classes, objects, and arrays to keep track of books, issue and return transactions, and availability status, thereby replacing the traditional manual methods of managing library resources. The project enables librarians and users to perform essential activities such as adding new books, searching for books, issuing and returning books, and maintaining a record of transactions with ease and accuracy.

The primary objective is to create a user-friendly system that automates key library functions, reduces paperwork, and improves data reliability. The system demonstrates fundamental OOP principles by encapsulating book details and transaction logic within classes and managing collections of book objects using arrays. This project helps users understand the real-world application of OOP concepts in managing inventory and transaction workflows effectively.

By implementing this system, the library operations become streamlined, minimizing errors in record-keeping and enhancing accessibility to library resources for both the staff and library members. Future improvements may include adding graphical user interface features and database integration for scalability.

Table of Contents

Introduction

A Library Management System (LMS) is an essential tool designed to automate and simplify the everyday operations of a library. It helps in managing the book inventory, tracking issuance and returns, and maintaining records of library users efficiently. The purpose of this project is to build a mini library management system using Object-Oriented Programming concepts such as classes, objects, and arrays to demonstrate a basic and functional system for managing book lists and transactions.

The system aims to reduce manual work involved in keeping records, searching books, and handling book issues and returns. By implementing this system, librarians and users can easily navigate the library's collection, manage the borrowing process, and ensure proper record-keeping with minimal errors. This project serves as a practical example of how OOP principles can be applied in real-world applications, facilitating better data organization and code manageability.

The project is motivated by the need for an easy-to-use, reliable software solution for managing small to medium-sized library operations, while also providing hands-on experience with fundamental programming concepts such as arrays for storing multiple book records and methods to handle book lending transactions.

Objectives

- To design a simple, user-friendly library system using Object-Oriented Programming
- Manage book list using arrays and objects for efficient storage
- Implement basic borrowing and returning of books functionality
- Demonstrate class and object interactions in Java (or chosen language)
- Develop understanding of arrays for managing multiple book records

Classes

- Description of main classes and their roles:
    - Book Class: attributes like bookId, title, author, availability status
    - Library Class: manages array of Book objects, operations for add, search, issue, return
    - User or Transaction Class (optional): to handle user interactions or issuing/returning process
- UML Class diagram or simple class description and relationships

Core Components and Code Snippets

- Book.java

```java
/**
 * Represents an individual book in the library with details.
 */
public class Book {
    private int id;                      // Unique ID of the book
    private String title;                // Title of the book
    private String author;               // Author's name
    private int totalCopies;             // Total copies of this book
    private int availableCopies;         // Currently available copies

    /**
     * Constructor initializes book details.
     */
    public Book(int id, String title, String author, int totalCopies) {
        this.id = id;
        this.title = title;
        this.author = author;
        this.totalCopies = totalCopies;
        this.availableCopies = totalCopies;  // Initially, all copies are
available
    }

    // Getters...
    public int getId() { return id; }
    public String getTitle() { return title; }
    public String getAuthor() { return author; }
    public int getTotalCopies() { return totalCopies; }
    public int getAvailableCopies() { return availableCopies; }

    /**
     * Issues a book if available.
     * @return true if book issued successfully, false if no copies available
     */
    public boolean issueBook() {
        if (availableCopies > 0) {
            availableCopies--;
            return true;
        }
        return false;
    }

    /**
     * Returns a book and updates availability.
     */
    public void returnBook() {
        if (availableCopies < totalCopies) {
            availableCopies++;
        }
    }
}
```

- BooksCollection.java

```java
import java.util.ArrayList;
import java.util.List;

/**
 * Manages collection of Book objects and related operations.
 */
```

```java
public class BooksCollection {
    private List<Book> books;

    public BooksCollection() {
        books = new ArrayList<>();
    }

    /**
     * Adds a new book to the collection.
     */
    public void addBook(Book book) {
        books.add(book);
        System.out.println("Book added: " + book.getTitle());
    }

    /**
     * Lists all books with available copies.
     */
    public void listBooks() {
        System.out.println("Library Books:");
        for (Book book : books) {
            System.out.printf("ID: %d | Title: %s | Author: %s | Available:
%d/%d%n",
                    book.getId(), book.getTitle(), book.getAuthor(),
                    book.getAvailableCopies(), book.getTotalCopies());
        }
    }

    /**
     * Issues a book by ID if available.
     */
    public boolean issueBook(int id) {
        for (Book book : books) {
            if (book.getId() == id) {
                if (book.issueBook()) {
                    System.out.println("Book issued: " + book.getTitle());
                    return true;
                } else {
                    System.out.println("Book '" + book.getTitle() + "' is
currently not available.");
                    return false;
                }
            }
        }
        System.out.println("Book ID " + id + " not found.");
        return false;
    }

    /**
     * Returns a book by ID.
     */
    public boolean returnBook(int id) {
        for (Book book : books) {
            if (book.getId() == id) {
                book.returnBook();
                System.out.println("Book returned: " + book.getTitle());
                return true;
            }
        }
        System.out.println("Book ID " + id + " not found.");
        return false;
    }
}
```

-
- Student.java

-
  ```java
  /**
   * Represents a student/user of the library.
   */
  public class Student {
      private int studentId;
      private String name;

      /**
       * Constructor initializes student details.
       */
      public Student(int studentId, String name) {
          this.studentId = studentId;
          this.name = name;
      }

      public int getStudentId() { return studentId; }
      public String getName() { return name; }
  }
  ```
-
- StudentsCollection.java

-
  ```java
  import java.util.ArrayList;
  import java.util.List;

  /**
   * Manages collection of Student objects.
   */
  public class StudentsCollection {
      private List<Student> students;

      public StudentsCollection() {
          students = new ArrayList<>();
      }

      /**
       * Adds a new student to the collection.
       */
      public void addStudent(Student student) {
          students.add(student);
          System.out.println("Student registered: " + student.getName());
      }

      /**
       * Finds a student by ID.
       */
      public Student findStudentById(int studentId) {
          for (Student student : students) {
              if (student.getStudentId() == studentId) {
                  return student;
              }
          }
          return null;
      }

      /**
       * Lists all registered students.
       */
      public void listStudents() {
          System.out.println("Registered Students:");
          for (Student student : students) {
              System.out.printf("ID: %d | Name: %s%n", student.getStudentId(),
  student.getName());
          }
      }
  }
  ```
-

- LibraryManagementApp.java

```java
import java.util.Scanner;

/**
 * Main class with menu-driven application flow.
 * Coordinates library book and student operations.
 */
public class LibraryManagementApp {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        BooksCollection books = new BooksCollection();
        StudentsCollection students = new StudentsCollection();

        while (true) {
            System.out.println("\nLibrary Management Menu:");
            System.out.println("1. Add Book");
            System.out.println("2. List Books");
            System.out.println("3. Add Student");
            System.out.println("4. List Students");
            System.out.println("5. Issue Book");
            System.out.println("6. Return Book");
            System.out.println("7. Exit");
            System.out.print("Enter choice: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    System.out.print("Book ID: ");
                    int bookId = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Book Title: ");
                    String title = scanner.nextLine();
                    System.out.print("Author: ");
                    String author = scanner.nextLine();
                    System.out.print("Total Copies: ");
                    int copies = scanner.nextInt();
                    scanner.nextLine();

                    Book book = new Book(bookId, title, author, copies);
                    books.addBook(book);
                    break;

                case 2:
                    books.listBooks();
                    break;

                case 3:
                    System.out.print("Student ID: ");
                    int studentId = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Student Name: ");
                    String name = scanner.nextLine();

                    Student student = new Student(studentId, name);
                    students.addStudent(student);
                    break;

                case 4:
                    students.listStudents();
                    break;

                case 5:
                    System.out.print("Enter Book ID to issue: ");
                    int issueBookId = scanner.nextInt();
                    scanner.nextLine();
                    System.out.print("Enter Student ID: ");
                    int issueStudentId = scanner.nextInt();
                    scanner.nextLine();

                    Student issueStudent =
students.findStudentById(issueStudentId);
                    if (issueStudent == null) {
                        System.out.println("Student not registered.");
                    } else {
                        books.issueBook(issueBookId);
                    }
                    break;

                case 6:
                    System.out.print("Enter Book ID to return: ");
                    int returnBookId = scanner.nextInt();
                    scanner.nextLine();

                    books.returnBook(returnBookId);
                    break;

                case 7:
                    System.out.println("Exiting Library Management System.");
                    scanner.close();
                    System.exit(0);

                default:
                    System.out.println("Invalid choice, try again.");
            }
        }
    }
}
```

-

Transaction Handling

Transaction Handling

The transaction handling page details the key processes involved in issuing and returning books in the Library Management System. It outlines how the system manages these tasks using classes, objects, and arrays to ensure smooth operation.

Transaction Handling in the Library Management System

1. Issue Book Process

- The user (member/librarian) initiates the issuing process by entering the Book ID and Member ID.
- The system searches the book array to find the requested book and checks if it is available (not issued).
- If available, the book's status is updated to "issued" and linked to the member's record.
- The issue date is recorded along with member details in the transactions array.
- Confirmation of successful issue is displayed; otherwise, an error message appears if the book is not found or unavailable.

2. Return Book Process

- The user starts the return process by providing the Book ID and Member ID.
- The system confirms if the book is currently issued to that member.
- Upon validation, the book's status is reset to "available" in the book array.
- The return date is recorded, and the transaction is updated in the transactions array.
- The system calculates any overdue fines based on return date (optional feature).
- Successful return confirmation is shown or an error message if the return is invalid.

3. Error Handling

- The system prevents issuing a book that is already issued.
- Return attempts for non-issued or unrecognized book-member pairs trigger appropriate errors.
- Input validation ensures Book ID and Member ID are in correct formats.

4. Transaction Flow Summary

- The book and transaction arrays serve as the data structures managing book availability and issue/return records.
- Object-oriented methods handle searching, updating, and validating transactions to maintain data integrity.
- The system provides real-time feedback on transaction results to guide users effectively.

5. Sample Pseudocode for Issue Book

text

```
function issueBook(bookId, memberId) {
 book = findBook(bookId)
 if (book == null) {
  return "Book not found"
  }
  if (book.status == ‘issued’) {
    return “Book already issued”
  }
  book.status = ‘issued’
  recordTransaction(bookId, memberId, issueDate = currentDate)
  return “Book issued successfully”
}
```

6. Sample Pseudocode for Return Book

```
text
function returnBook(bookId, memberId) {
  transaction = findTransaction(bookId, memberId)
  if (transaction == null or transaction.status == ‘returned’) {
    return “Invalid return”
  }
  book = findBook(bookId)
  book.status = ‘available’
  transaction.status = ‘returned’
  transaction.returnDate = currentDate
  calculateFine(transaction.issueDate, currentDate) // optional
  return “Book returned successfully”
}
```

Conclusion and Team Contribution

The project successfully simulates essential library management operations through a simple, console-based Java application. It demonstrates the practical application of classes, objects, arrays, and basic OOP principles to efficiently manage a book collection, including issuing and returning books. The system effectively organizes book data and handles transactions with clarity and modularity.

Team Contributions:

- Kashish – Designed and implemented the Book class and managed book list storage using arrays.
- Preet (Pilwan) – Developed core functionality for book issuance and return processes including validation.
- Tanishka – Created the main menu interface and handled user input validation to ensure smooth interactions.
- Sanjal – Conducted testing and debugging to enhance application stability and reliability.
- Ritika – Prepared project documentation and presentation materials, and coordinated team efforts.