PES University

Department of Computer Science and Engineering

Python for Computational Problem Solving Lab

Hackathon - Jackfruit Level Problem solving

Date: 02/01/2025

Team Details:

Team 5

Srivani   : PES1UG24CS471
Spoorthi P : PES1UG24AM279
Suraj HP : PES1UG24EC221
Srijan Akshit : PES1UG24AM284
Srijan S K: PES1UG24EC214

Class: P7

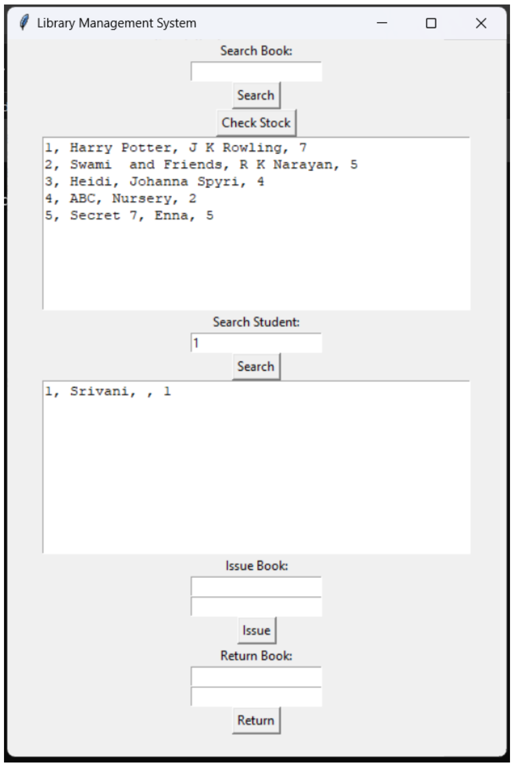
Code:

```
import csv
from datetime import datetime, timedelta
import tkinter as tk
from tkinter import ttk, messagebox

class Book:
    def __init__(self, book_id, title, author, available_copies):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.available_copies = int(available_copies)

    def to_csv(self):
        return f"{self.book_id},{self.title},{self.author},{self.available_copies}"

class Student:
    def __init__(self, student_id, name):
        self.student_id = student_id
        self.name = name
        self.borrowed_books = []

    def add_borrowed_book(self, book_id):
        self.borrowed_books.append(book_id)

    def remove_borrowed_book(self, book_id):
        if book_id in self.borrowed_books:
            self.borrowed_books.remove(book_id)

    def to_csv(self):
        return f"{self.student_id},{self.name},{','.join(self.borrowed_books)}"

    def is_eligible(self):
        return len(self.borrowed_books) < 3  # Assuming a student can borrow up to 3 books


class Librarian:
    def __init__(self):
        self.books = []
        self.students = []
        self.logs = []
        self.load_data()

    def load_data(self):
        with open('books.csv', 'r') as file:
            reader = csv.reader(file)
            next(reader)  # Skip header
            self.books = [Book(row[0], row[1], row[2], row[3]) for row in reader]

        with open('students.csv', 'r') as file:
            reader = csv.reader(file)
            next(reader)  # Skip header
            self.students = []
            for row in reader:
                student_id, name, *borrowed_books = row
                student = Student(student_id, name)
                student.borrowed_books = borrowed_books  # Load borrowed books correctly
                self.students.append(student)

        with open('logs.csv', 'r') as file:
            reader = csv.reader(file)
            next(reader)  # Skip header
            self.logs = [row for row in reader]


    def save_data(self):
        with open('books.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(['book_id', 'title', 'author', 'available_copies'])
            writer.writerows([book.to_csv().split(',') for book in self.books])

        with open('students.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(['student_id', 'name', 'borrowed_books'])
            writer.writerows([student.to_csv().split(',') for student in self.students])

        with open('logs.csv', 'w', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(['transaction_id', 'transaction_type', 'book_id', 'student_id', 'issue_date',
'return_date', 'penalty'])
            writer.writerows(self.logs)
    def check_stock(self):
        return [book for book in self.books if book.available_copies > 0]

    def search_book(self, query):
        return [book for book in self.books if query in (book.title, book.author, str(book.book_id))]

    def search_student(self, query):
        return [student for student in self.students if query in (student.student_id, student.name)]

    def issue_book(self, book_id, student_id):
        book = next((b for b in self.books if b.book_id == book_id), None)
        student = next((s for s in self.students if s.student_id == student_id), None)
        if book and student and book.available_copies > 0 and student.is_eligible():
            book.available_copies -= 1
            student.add_borrowed_book(book_id)
            issue_date = datetime.now().strftime('%m/%d/%Y')
            self.update_logs('issue', book_id, student_id, issue_date)
            self.save_data()
            return True
        return False

    def return_book(self, book_id, student_id):
        book = next((b for b in self.books if b.book_id == book_id), None)
        student = next((s for s in self.students if s.student_id == student_id), None)
        if book and student and book_id in student.borrowed_books:
            book.available_copies += 1
            student.remove_borrowed_book(book_id)
            return_date = datetime.now().strftime('%m/%d/%Y')
            issue_date = next((log[4] for log in self.logs if log[2] == book_id and log[3] == student_id
and log[1] == 'issue'), None)
            penalty = self.calculate_penalty(issue_date)
            self.update_logs('return', book_id, student_id, issue_date, return_date, penalty)
            self.save_data()
            return penalty
        return None

    def calculate_penalty(self, issue_date):
        issue_date = datetime.strptime(issue_date, '%m/%d/%Y')
        print(issue_date)
        return_date = datetime.now()
        days_late = (return_date - issue_date).days - 2
        if days_late > 0:
            return min(days_late * 1, 50)
        return 0

    def update_logs(self, transaction_type, book_id, student_id, issue_date, return_date=None, 
penalty=0):
        transaction_id = len(self.logs) + 1
        self.logs.append([transaction_id, transaction_type, book_id, student_id, issue_date, 
return_date, penalty])

class LibraryApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Library Management System")
        self.librarian = Librarian()
        self.create_widgets()

    def create_widgets(self):
        # Book Management Section
        tk.Label(self.root, text="Search Book:").pack()
        self.book_search_entry = tk.Entry(self.root)
        self.book_search_entry.pack()
        tk.Button(self.root, text="Search", command=self.search_book).pack()
        tk.Button(self.root, text="Check Stock", command=self.check_stock).pack()
        self.book_results = tk.Text(self.root, height=10, width=50)
        self.book_results.pack()

        # Student Management Section
        tk.Label(self.root, text="Search Student:").pack()
        self.student_search_entry = tk.Entry(self.root)
        self.student_search_entry.pack()
        tk.Button(self.root, text="Search", command=self.search_student).pack()
        self.student_results = tk.Text(self.root, height=10, width=50)
        self.student_results.pack()

        # Transaction Section
        tk.Label(self.root, text="Issue Book:").pack()
        self.issue_book_id_entry = tk.Entry(self.root)
        self.issue_book_id_entry.pack()
        self.issue_student_id_entry = tk.Entry(self.root)
        self.issue_student_id_entry.pack()
        tk.Button(self.root, text="Issue", command=self.issue_book).pack()
        tk.Label(self.root, text="Return Book:").pack()
        self.return_book_id_entry = tk.Entry(self.root)
        self.return_book_id_entry.pack()
        self.return_student_id_entry = tk.Entry(self.root)
        self.return_student_id_entry.pack()
        tk.Button(self.root, text="Return", command=self.return_book).pack()
        self.penalty_label = tk.Label(self.root, text="")
        self.penalty_label.pack()

    def search_book(self):
        query = self.book_search_entry.get()
        results = self.librarian.search_book(query)
        self.book_results.delete(1.0, tk.END)
        if results:
            for book in results:
                self.book_results.insert(tk.END,    f"{book.book_id},    {book.title},    {book.author},
{book.available_copies}\n")
        else:
            self.book_results.insert(tk.END, "No results found.")

    def check_stock(self):
        results = self.librarian.check_stock()
        self.book_results.delete(1.0, tk.END)
        if results:
            for book in results:
                self.book_results.insert(tk.END,    f"{book.book_id},    {book.title},    {book.author},
{book.available_copies}\n")
        else:
            self.book_results.insert(tk.END, "No books available.")

    def search_student(self):
        query = self.student_search_entry.get()
        results = self.librarian.search_student(query)
        self.student_results.delete(1.0, tk.END)
        if results:
            for student in results:
                self.student_results.insert(tk.END,    f"{student.student_id},    {student.name},    {',
'.join(student.borrowed_books)}\n")
        else:
            self.student_results.insert(tk.END, "No results found.")
```

Python for Computational Problem Solving Lab

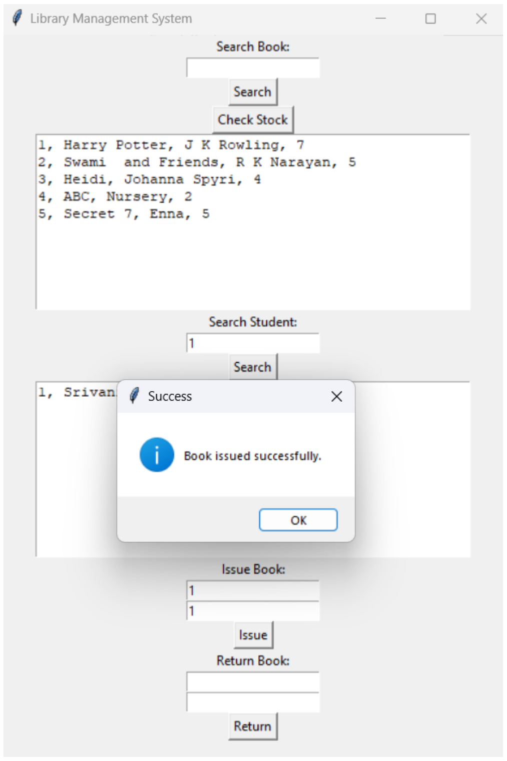
```
    def issue_book(self):
        book_id = self.issue_book_id_entry.get().strip()
        student_id = self.issue_student_id_entry.get().strip()

        if not book_id or not student_id:
            messagebox.showerror("Input Error", "Please enter both Book ID and Student ID.")
            return

        try:
            success = self.librarian.issue_book(book_id, student_id)
            if success:
                messagebox.showinfo("Success", "Book issued successfully.")
                self.issue_book_id_entry.delete(0, tk.END)
                self.issue_student_id_entry.delete(0, tk.END)
            else:
                messagebox.showerror("Error", "Failed to issue book. Please check the details.")
        except PermissionError as e:
            messagebox.showerror("Permission Error", f"Permission error: {e}")
        except Exception as e:
            messagebox.showerror("Error", f"An unexpected error occurred: {e}")


    def return_book(self):
        book_id = self.return_book_id_entry.get()
        student_id = self.return_student_id_entry.get()
        penalty = self.librarian.return_book(book_id, student_id)
        if penalty is not None:
            messagebox.showinfo("Success", f"Book returned successfully. Penalty: ${penalty}")
            self.penalty_label.config(text=f"Penalty: ${penalty}")
        else:
            messagebox.showerror("Error", "Failed to return book.")

if __name__ == "__main__":
    root = tk.Tk()
    app = LibraryApp(root)
    root.mainloop()
```

Output (GUI) Screenshots:

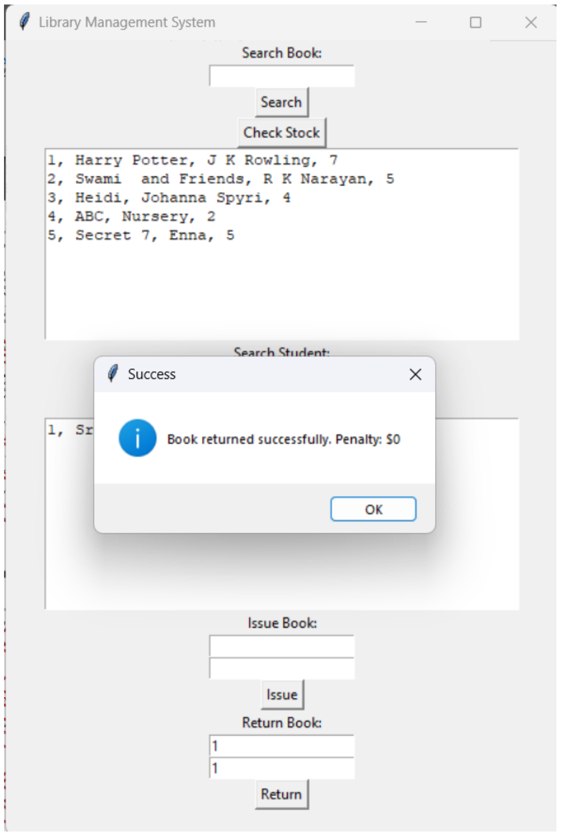


```
1, Harry Potter, J K Rowling, 7
2, Swami  and Friends, R K Narayan, 5
3, Heidi, Johanna Spyri, 4
4, ABC, Nursery, 2
5, Secret 7, Enna, 5
```

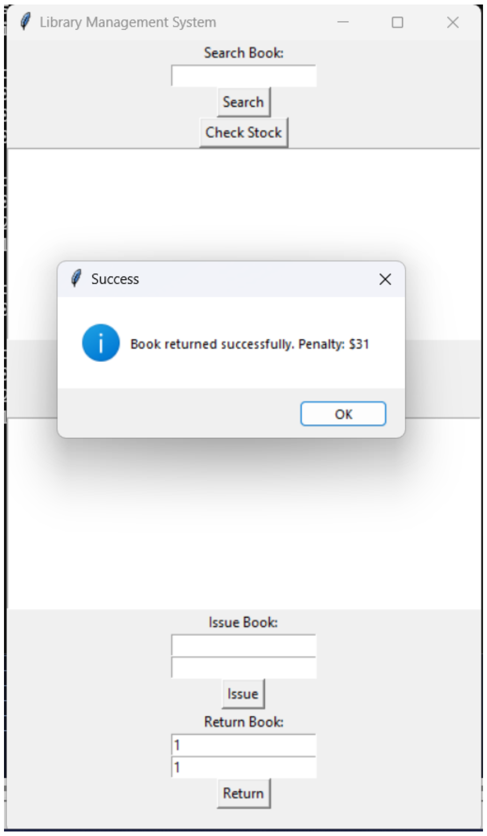
```
1, Srivani, , 1
```



Book issued successfully.



Book returned successfully. Penalty: $0



Book returned successfully. Penalty: $31