**E – Library System**

import json

import os

import logging

from datetime import datetime, timedelta

from typing import Dict, List

STORE = "library.json"

LOGFILE = "library.log"

logging.basicConfig(filename=LOGFILE, level=logging.INFO, format="%(asctime)s %(levelname)s: %(message)s")

LOAN\_DAYS = 14

FINE\_PER\_DAY = 2

class Book:

def \_\_init\_\_(self, book\_id: str, title: str, author: str, isbn: str, available: bool = True):

self.book\_id = book\_id

self.title = title

self.author = author

self.isbn = isbn

self.available = available

def as\_dict(self):

return {

"book\_id": self.book\_id,

"title": self.title,

"author": self.author,

"isbn": self.isbn,

"available": self.available,

}

@staticmethod

def from\_dict(d: dict) -> "Book":

return Book(d["book\_id"], d["title"], d["author"], d["isbn"], d.get("available", True))

class User:

def \_\_init\_\_(self, user\_id: str, name: str):

self.user\_id = user\_id

self.name = name

def as\_dict(self):

return {"user\_id": self.user\_id, "name": self.name}

@staticmethod

def from\_dict(d: dict) -> "User":

return User(d["user\_id"], d["name"])

class LibrarySystem:

def \_\_init\_\_(self):

self.books: Dict[str, Book] = {}

self.users: Dict[str, User] = {}

self.transactions: List[dict] = []

self.\_load()

def \_load(self):

if not os.path.exists(STORE):

self.\_save()

return

with open(STORE, "r", encoding="utf-8") as f:

data = json.load(f)

self.books = {b["book\_id"]: Book.from\_dict(b) for b in data.get("books", [])}

self.users = {u["user\_id"]: User.from\_dict(u) for u in data.get("users", [])}

self.transactions = data.get("transactions", [])

def \_save(self):

data = {

"books": [b.as\_dict() for b in self.books.values()],

"users": [u.as\_dict() for u in self.users.values()],

"transactions": self.transactions,

}

with open(STORE, "w", encoding="utf-8") as f:

json.dump(data, f, indent=2)

def add\_book(self, book\_id: str, title: str, author: str, isbn: str):

if book\_id in self.books:

raise ValueError("Book ID exists")

self.books[book\_id] = Book(book\_id, title, author, isbn)

self.\_save()

logging.info(f"Book added: {book\_id} - {title}")

def remove\_book(self, book\_id: str):

b = self.books.get(book\_id)

if not b:

raise LookupError("Book not found")

if not b.available:

raise RuntimeError("Book is currently issued")

del self.books[book\_id]

self.\_save()

logging.info(f"Book removed: {book\_id}")

def register\_user(self, user\_id: str, name: str):

if user\_id in self.users:

raise ValueError("User ID exists")

self.users[user\_id] = User(user\_id, name)

self.\_save()

logging.info(f"User registered: {user\_id} - {name}")

def issue\_book(self, user\_id: str, book\_id: str):

user = self.users.get(user\_id)

book = self.books.get(book\_id)

if not user:

raise LookupError("User not found")

if not book:

raise LookupError("Book not found")

if not book.available:

raise RuntimeError("Book already issued")

issue\_date = datetime.now()

due\_date = issue\_date + timedelta(days=LOAN\_DAYS)

self.transactions.append({

"book\_id": book\_id,

"user\_id": user\_id,

"issue\_date": issue\_date.isoformat(),

"due\_date": due\_date.isoformat(),

"return\_date": None,

})

book.available = False

self.\_save()

logging.info(f"Issued {book\_id} to {user\_id}")

def give\_back\_book(self, user\_id: str, book\_id: str) -> int:

book = self.books.get(book\_id)

if not book:

raise LookupError("Book not found")

tx = None

for t in self.transactions:

if t["book\_id"] == book\_id and t["user\_id"] == user\_id and t["return\_date"] is None:

tx = t

break

if tx is None:

raise LookupError("Active transaction not found")

now = datetime.now()

tx["return\_date"] = now.isoformat()

book.available = True

self.\_save()

due = datetime.fromisoformat(tx["due\_date"])

late\_days = max(0, (now.date() - due.date()).days)

fine = late\_days \* FINE\_PER\_DAY

if fine:

logging.info(f"Fine for {user\_id} on {book\_id}: {fine}")

return fine

def list\_books(self):

for b in self.books.values():

status = "Available" if b.available else "Issued"

print(f"[{b.book\_id}] {b.title} / {b.author} / {b.isbn} — {status}")

def list\_users(self):

for u in self.users.values():

print(f"[{u.user\_id}] {u.name}")

def list\_active\_loans(self):

for t in self.transactions:

if t["return\_date"] is None:

print(f"Book {t['book\_id']} -> User {t['user\_id']} | Due {t['due\_date']}")

def run\_cli():

lib = LibrarySystem()

MENU = """

e-Library System

1) Add Book

2) Remove Book

3) Register User

4) Issue Book

5) Return Book

6) List Books

7) List Users

8) Active Loans

0) Exit

Choice: """

while True:

try:

choice = input(MENU).strip()

if choice == "1":

lib.add\_book(input("Book ID: "), input("Title: "), input("Author: "), input("ISBN: "))

elif choice == "2":

lib.remove\_book(input("Book ID: "))

elif choice == "3":

lib.register\_user(input("User ID: "), input("Name: "))

elif choice == "4":

lib.issue\_book(input("User ID: "), input("Book ID: "))

elif choice == "5":

fine = lib.give\_back\_book(input("User ID: "), input("Book ID: "))

print(f"Returned. Fine: {fine}")

elif choice == "6":

lib.list\_books()

elif choice == "7":

lib.list\_users()

elif choice == "8":

lib.list\_active\_loans()

elif choice == "0":

break

else:

print("Invalid choice.")

except Exception as e:

print("Error:", e)

if \_\_name\_\_ == "\_\_main\_\_":

run\_cli()

pass