21. Force the system to handle simultaneous borrow attempts (simulate concurrency with threading).

We can use Python’s threading module. Each borrow attempt runs in a separate thread. To prevent data corruption, protect shared resources like the books list or file writes with threading.Lock.

22. Implement input validation: member ID must be alphanumeric, book ID must be unique.

We have to use str.isalnum() to check member IDs.

class Library:

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

if book\_id in self.books:

print("Book ID already exists.")

return

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

self.save()

print("Book added.")

def register\_member(self, member\_id: str, name: str):

if not member\_id.isalnum():

print("Member ID must be alphanumeric.")

return

if member\_id in self.members:

print("Member ID already exists.")

return

self.members[member\_id] = Member(member\_id, name)

self.save()

print("Member registered.")

23. Raise an exception if a member tries to borrow more than 5 books at once.

We have to raise a custom exception. Before borrowing we should count how many books the member already has. If ≥ 5, block the action.

class Library:

def borrow\_book(self, member\_id: str, book\_id: str):

with self.lock:

m = self.members.get(member\_id)

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

if not m:

print("Member not found.")

return

if len(m.borrowed) >= 5:

raise Exception(f"{m.name} already borrowed 5 books.")

if not b:

print("Book not found.")

return

if not b.available:

print("Book already issued.")

return

b.available = False

m.borrowed.append(book\_id)

self.save()

print(f"{m.name} borrowed '{b.title}'.")

24. Implement retry logic if the file is locked when saving.

Loop with try/except and time.sleep().

import time

class Library:

def save(self):

for attempt in range(3):

try:

with open(BOOKS\_FILE, "w", encoding="utf-8") as f:

for b in self.books.values():

f.write(b.to\_line())

with open(MEMBERS\_FILE, "w", encoding="utf-8") as f:

for m in self.members.values():

f.write(m.to\_line())

return

except PermissionError:

print("File locked, retrying...")

time.sleep(1)

print("Failed to save after 3 attempts.")

D. Persistence & File/JSON

25. Add versioning to the JSON file, so each save creates a backup copy.

Before saving we have to copy the old file as library\_backup\_TIMESTAMP.json and use shutil.copy().

code->

import json

import shutil

from datetime import datetime

class Library:

def save\_json\_versioned(self):

data = {

"books": [vars(b) for b in self.books.values()],

"members": [vars(m) for m in self.members.values()]

}

with open("library.json", "w", encoding="utf-8") as f:

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

ts = datetime.now().strftime("%Y%m%d\_%H%M%S")

shutil.copy("library.json", f"library\_{ts}.json")

print("Library saved with versioning.")

26. Use with open(..., 'a') to implement an append-only log file for all actions.

class Library:

def log\_action(self, msg: str):

with open("library.log", "a", encoding="utf-8") as f:

f.write(f"{datetime.now()} - {msg}\n")

def borrow\_book(self, member\_id: str, book\_id: str):

with self.lock:

self.log\_action(f"{member\_id} borrowed {book\_id}")

27. Add an import/export feature (JSON ↔ TXT ↔ CSV).

JSON → TXT/CSV: Parse JSON and write rows using csv.writer or plain text.

TXT/CSV → JSON: Read line by line, then dump to JSON using json.dump().

import csv

class Library:

def export\_csv(self):

with open("library\_export.csv", "w", newline='', encoding="utf-8") as f:

writer = csv.writer(f)

writer.writerow(["BookID", "Title", "Author", "ISBN", "Available"])

for b in self.books.values():

writer.writerow([b.book\_id, b.title, b.author, b.isbn, b.available])

def import\_json(self, file="library.json"):

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

data = json.load(f)

for b in data["books"]:

self.books[b["book\_id"]] = Book(\*\*b)

for m in data["members"]:

self.members[m["member\_id"]] = Member(\*\*m)

28. Store the last modified timestamp of each book inside the JSON data.

from datetime import datetime

While updating book data (borrow/return),

add book["last\_modified"] = datetime.now().isoformat().

class Book:

def \_\_init\_\_(self, book\_id, title, author, isbn, available=True, last\_modified=None):

self.book\_id = book\_id

self.title = title

self.author = author

self.isbn = isbn

self.available = available

self.last\_modified = last\_modified or datetime.now().isoformat()

def update\_timestamp(self):

self.last\_modified = datetime.now().isoformat()

class Library:

def borrow\_book(self, member\_id, book\_id):

with self.lock:

b.update\_timestamp()

29. Use pickle for faster serialization of the entire library state.

import pickle

class Library:

def save\_pickle(self):

with open("library.pkl", "wb") as f:

pickle.dump(self, f)

@staticmethod

def load\_pickle():

with open("library.pkl", "rb") as f:

return pickle.load(f)

E. Date & Time / Business Logic

30. Implement a fine calculator that charges different rates based on how late a book is returned (sliding scale).

class Library:

def calculate\_fine(self, days\_late):

if days\_late <= 0:

return 0

elif days\_late <= 5:

return days\_late \* 1

elif days\_late <= 10:

return 5 + (days\_late-5)\*2

else:

return 15 + (days\_late-10)\*5