**Day-23 Morning Assessment**

21. import threading  
  
lock = threading.Lock()  
  
class Library:  
   \_instance = None  
  
   def \_\_new\_\_(cls):  
       if cls.\_instance is None:  
           cls.\_instance = super().\_\_new\_\_(cls)  
           cls.\_instance.books = []  
           cls.\_instance.members = []  
       return cls.\_instance  
  
   def borrow\_book(self, member, book):  
       with lock:  # ensures only one thread modifies at a time  
           if not book.isAvailable:  
               raise BookNotAvailableError(f"{book.title} already borrowed!")  
           if len(member.borrowed\_books) >= 5:  
               raise Exception("Cannot borrow more than 5 books!")  
           book.isAvailable = False  
           member.borrowed\_books.append(book)

Then simulate:  
thread1 = threading.Thread(target=library.borrow\_book, args=(m1, b1))  
thread2 = threading.Thread(target=library.borrow\_book, args=(m2, b1))  
  
thread1.start()  
thread2.start()  
thread1.join()  
thread2.join()  
  
22. class Member:  
   def \_\_init\_\_(self, name, member\_id):  
       if not member\_id.isalnum():  
           raise ValueError("Member ID must be alphanumeric")  
       self.name = name  
       self.member\_id = member\_id  
       self.borrowed\_books = []  
  
class Library:  
   ...  
   def add\_book(self, book):  
       if any(b.isbn == book.isbn for b in self.books):  
           raise ValueError("Book ISBN must be unique")  
       self.books.append(book)  
       return self  
  
23. class BorrowLimitExceededError(Exception):  
   def \_\_init\_\_(self, message="Cannot borrow more than 5 books"):  
       super().\_\_init\_\_(message)

Modify borrow:  
if len(member.borrowed\_books) >= 5:  
   raise BorrowLimitExceededError()  
  
24. import time, os  
  
def save\_with\_retry(filename, data, retries=3, delay=1):  
   for attempt in range(retries):  
       try:  
           with open(filename, "w") as f:  
               f.write(data)  
           break  
       except PermissionError:  
           if attempt < retries - 1:  
               time.sleep(delay)  
           else:  
               raise

**D. Persistence & File/JSON**  
  
25. import shutil, datetime, json  
  
def save\_library(library, filename="library.json"):  
   if os.path.exists(filename):  
       timestamp = datetime.datetime.now().strftime("%Y%m%d%H%M%S")  
       shutil.copy(filename, f"library\_backup\_{timestamp}.json")  
  
   with open(filename, "w") as f:  
       json.dump([book.\_\_dict\_\_ for book in library.books], f, indent=4)  
  
26. def log\_action(action):  
   with open("library\_actions.log", "a") as f:  
       f.write(f"{datetime.datetime.now()} - {action}\n")  
  
27. import csv  
  
def export\_books\_to\_csv(library, filename="books.csv"):  
   with open(filename, "w", newline="") as f:  
       writer = csv.DictWriter(f, fieldnames=["title", "author", "isbn", "isAvailable"])  
       writer.writeheader()  
       writer.writerows([book.\_\_dict\_\_ for book in library.books])  
  
def import\_books\_from\_csv(filename="books.csv"):  
   books = []  
   with open(filename, "r") as f:  
       reader = csv.DictReader(f)  
       for row in reader:  
           books.append(Book(row["title"], row["author"], row["isbn"], row["isAvailable"] == "True"))  
   return books

28. class Book:  
   def \_\_init\_\_(self, title, author, isbn):  
       self.title = title  
       self.author = author  
       self.isbn = isbn  
       self.isAvailable = True  
       self.last\_modified = datetime.datetime.now()  
  
   def update\_timestamp(self):  
       self.last\_modified = datetime.datetime.now()  
  
29. import pickle  
  
def save\_pickle(library, filename="library.pkl"):  
   with open(filename, "wb") as f:  
       pickle.dump(library, f)  
  
def load\_pickle(filename="library.pkl"):  
   with open(filename, "rb") as f:  
       return pickle.load(f)

**E. Date & Time/Business Logic**

30. def calculate\_fine(days\_late):  
   if days\_late <= 0:  
       return 0  
   elif days\_late <= 5:  
       return days\_late \* 2     
   elif days\_late <= 10:  
       return (5 \* 2) + (days\_late - 5) \* 5

   else:  
       return (5 \* 2) + (5 \* 5) + (days\_late - 10) \* 10