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

```
import threading
lock = threading.Lock()
def issue_book(self, book_id, user_id):
    with lock:
        if book_id not in self.books:
            raise LookupError("Book not found")
        book = self.books[book_id]
        if not book.available:
            raise BookNotAvailableError("Book already issued")
        book.available = False
        self.transactions.append({"book": book_id, "user": user_id, "date": datetime.now().isoformat()})
        self._save()
```

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

```
import re
def register_user(self, user_id, name):
    if not re.match("^[A-Za-z0-9]+$", user_id):
        raise ValueError("Member ID must be alphanumeric")
    if user_id in self.users:
        raise ValueError("User already exists")
    self.users[user_id] = User(user_id, name)
    self._save()
def add_book(self, book_id, title, author, isbn):
    if book_id in self.books:
        raise ValueError("Book ID must be unique")
    self.books[book_id] = Book(book_id, title, author, isbn)
    self._save()
```

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

```
def issue_book(self, book_id, user_id):
    borrowed = [t for t in self.transactions if t["user"] == user_id and "return_date" not in t]
    if len(borrowed) >= 5:
        raise Exception("Member cannot borrow more than 5 books")
```

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

```
import time
def _save(self):
    retries = 3
    for i in range(retries):
        try:
            with open(STORE, "w", encoding="utf-8") as f:
                data = {
                    "books": [b.__dict__ for b in self.books.values()],
                    "users": [u.__dict__ for u in self.users.values()],
                    "transactions": self.transactions
                }
                json.dump(data, f, indent=4)
            break
        except PermissionError:
            logging.warning("File locked, retrying...")
            time.sleep(1)
    else:
        logging.error("Failed to save after retries")
```

D. Persistence & File/JSON

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

```
import shutil
def _save(self):
    if os.path.exists(STORE):
        backup = f"{STORE}_{datetime.now().strftime('%Y%m%d%H%M%S')}.bak"
        shutil.copy(STORE, backup)
    with open(STORE, "w", encoding="utf-8") as f:
        data = {
            "books": [b.__dict__ for b in self.books.values()],
            "users": [u.__dict__ for u in self.users.values()],
            "transactions": self.transactions
        }
        json.dump(data, f, indent=4)
```

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

```
def log_action(self, action):
    with open("actions.log", "a", encoding="utf-8") as f:
        f.write(f"{datetime.now().isoformat()} - {action}\n")
self.log_action(f"Book issued: {book_id} by {user_id}")
```

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

```
import csv
def export_csv(self, filename="library.csv"):
    with open(filename, "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 export_txt(self, filename="library.txt"):
    with open(filename, "w", encoding="utf-8") as f:
        for b in self.books.values():
            f.write(f"{b.book_id} - {b.title} - {b.author}\n")
def import_json(self, filename="library.json"):
    with open(filename, "r", encoding="utf-8") as f:
        data = json.load(f)
        self.books = {b["book_id"]: Book(**b) for b in data["books"]}
```

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

```
class Book:
    def __init__(self, book_id, title, author, isbn, available=True):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.isbn = isbn
        self.available = available
        self.last_modified = datetime.now().isoformat()
    def update(self, title=None, author=None):
        if title: self.title = title
        if author: self.author = author
        self.last_modified = datetime.now().isoformat()
```

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

```
import pickle

def save_pickle(self, filename="library.pkl"):
    with open(filename, "wb") as f:
        pickle.dump(self, f)

@staticmethod

def load_pickle(filename="library.pkl"):
    with open(filename, "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).

```
def calculate_fine(self, borrow_date, return_date):
    days = (return_date - borrow_date).days
    if days <= 14:
        return 0
    late = days - 14
    if late <= 5:
        return late * 2
    elif late <= 10:
        return (5 * 2) + (late - 5) * 5
    else:
        return (5 * 2) + (5 * 5) + (late - 10) * 10
```