11. Use property decorators (@property) for available_books_count.

```
@property
def available_books_count(self) -> int:
    return sum(1 for b in self.books.values() if b.available)
```

12. Create an abstract base class (ABC) Person, inherited by Member and Librarian.

```
from abc import ABC, abstractmethod
class Person(ABC):
    def __init__(self, user_id, name):
        self.user_id = user_id
        self.name = name
    @abstractmethod
    def role(self):
        pass
class Member(Person):
    def role(self):
        return "Member"
class Librarian(Person):
    def role(self):
        return "Librarian"
```

13. Demonstrate multiple inheritance by creating a ResearchScholar who is both StudentMember and FacultyMember.

```
class StudentMember(Member):
    def role(self):
```

```
        return "Student Member"
class FacultyMember(Member):
    def role(self):
        return "Faculty Member"
class ResearchScholar(StudentMember, FacultyMember):
    def role(self):
        return "Research Scholar"
```

14. Override __str__ and __repr__ methods for clean debugging outputs.

```
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
    def __str__(self):
        return f"{self.title} by {self.author} (ISBN: {self.isbn})"
    def __repr__(self):
        return f"Book({self.book_id}, {self.title}, {self.author}, {self.isbn}, {self.available})"
```

15. Add a Singleton pattern for the Library class (only one instance should exist).

```
class LibrarySystem:
    _instance = None
```

```
    def __new__(cls, *args, **kwargs):
        if cls._instance is None:
            cls._instance = super(LibrarySystem, cls).__new__(cls)
        return cls._instance
```

16. Implement a Factory Method for creating different types of members (StudentMember, FacultyMember).

```
class MemberFactory:
    @staticmethod
    def create_member(member_type, user_id, name):
        if member_type == "student":
            return StudentMember(user_id, name)
        elif member_type == "faculty":
            return FacultyMember(user_id, name)
        else:
            return Member(user_id, name)
```

17. Add Method Chaining support, e.g., library.add_book(...).register_member(...).

```
def add_book(self, book_id, title, author, isbn):
    if book_id in self.books:
        raise ValueError("Book already exists")
    self.books[book_id] = Book(book_id, title, author, isbn)
    self._save()
    return self
```

18. Create a mixin class that adds to_json() and from_json() methods for books and members.

```
import json
class JsonMixin:
    def to_json(self):
        return json.dumps(self.__dict__)
    @classmethod
    def from_json(cls, data):
        return cls(**json.loads(data))
class Book(JsonMixin):
    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
```

C. Error Handling & Robustness

19. Write custom exception classes: BookNotAvailableError, MemberNotFoundError.

```
class BookNotAvailableError(Exception):
    pass
class MemberNotFoundError(Exception):
    pass
def issue_book(self, user_id: str, book_id: str):
    user = self.users.get(user_id)
    if not user:
```

```
        raise MemberNotFoundError("User not found")
    book = self.books.get(book_id)
    if not book:
        raise LookupError("Book not found")
    if not book.available:
        raise BookNotAvailableError("Book already issued")
```

20. Add a try-except-else-finally block around file reading/writing with proper error logging.

```
def _load(self):
    try:
        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(**b) for b in data["books"]}
        self.users = {u["user_id"]: User(**u) for u in data["users"]}
    except (json.JSONDecodeError, FileNotFoundError) as e:
        logging.error(f"Error loading data: {e}")
        self.books, self.users = {}, {}
    else:
        logging.info("Library data loaded successfully")
    finally:
        logging.info("Load operation finished")
```