**EVENING ASSIGNMENT**

**11. How can we use @property to calculate available books in the Library?**

class Library:

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

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

self.members: Dict[str, Member] = {}

@property

def available\_books\_count(self):

return sum(1 for book in self.books.values() if book.available)

**12. How can we create an abstract base class Person that is inherited by Member and Librarian?**

from abc import ABC, abstractmethod

class Person(ABC):

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

self.member\_id = member\_id

self.name = name

@abstractmethod

def get\_role(self):

pass

class Member(Person):

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

super().\_\_init\_\_(member\_id, name)

self.borrowed\_books = []

def get\_role(self):

return "Library Member"

class Librarian(Person):

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

super().\_\_init\_\_(member\_id, name)

def get\_role(self):

return "Librarian"

**13. How can we demonstrate multiple inheritance using a ResearchScholar who is both a StudentMember and FacultyMember?**

class StudentMember(Member):

def get\_role(self):

return "Student Member"

class FacultyMember(Member):

def get\_role(self):

return "Faculty Member"

class ResearchScholar(StudentMember, FacultyMember):

def get\_role(self):

return "Research Scholar (Student + Faculty)"

**14. How can we override \_\_str\_\_ and \_\_repr\_\_ for clean debugging outputs?**

class Book:

def \_\_str\_\_(self):

return f"Book[{self.book\_id}] {self.title} by {self.author} - {'Available' if self.available else 'Not Available'}"

def \_\_repr\_\_(self):

return f"Book(book\_id={self.book\_id}, title={self.title}, author={self.author}, available={self.available})"

**15. How can we apply the Singleton pattern to the Library class?**

class Library:

\_instance = None

def \_\_new\_\_(cls):

if cls.\_instance is None:

cls.\_instance = super(Library, cls).\_\_new\_\_(cls)

cls.\_instance.books = {}

cls.\_instance.members = {}

return cls.\_instance

**16. How can we use a Factory Method to create different types of members?**

class MemberFactory:

@staticmethod

def create\_member(member\_type: str, member\_id: str, name: str) -> Member:

if member\_type == "student":

return StudentMember(member\_id, name)

elif member\_type == "faculty":

return FacultyMember(member\_id, name)

elif member\_type == "scholar":

return ResearchScholar(member\_id, name)

else:

raise ValueError("Unknown member type")

**17. How can we implement method chaining in the Library class?**

class Library:

def add\_book(self, book: Book):

self.books[book.book\_id] = book

return self # enables chaining

def register\_member(self, member: Member):

self.members[member.member\_id] = member

return self

**18. How can we add JSON serialization using a Mixin class?**

import json

class JsonMixin:

def to\_json(self):

return json.dumps(self.\_\_dict\_\_)

@classmethod

def from\_json(cls, json\_str):

return cls(\*\*json.loads(json\_str))

class Book(JsonMixin):

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

self.book\_id = book\_id

self.title = title

self.author = author

self.available = available

**19. How can we write custom exception classes for handling errors?**

class BookNotAvailableError(Exception):

pass

class MemberNotFoundError(Exception):

pass

**Usage Example:**

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

if member\_id not in self.members:

raise MemberNotFoundError("Member does not exist!")

if not self.books[book\_id].available:

raise BookNotAvailableError("Book is not available for borrowing!")

**20. How can we use try-except-else-finally for safe file handling?**

def load(self):

try:

if os.path.exists("books.txt"):

with open("books.txt", "r") as f:

for line in f:

book = Book.from\_line(line)

self.books[book.book\_id] = book

if os.path.exists("members.txt"):

with open("members.txt", "r") as f:

for line in f:

member = Member.from\_line(line)

self.members[member.member\_id] = member

except Exception as e:

print(f"Error loading files: {e}")

else:

print("Files loaded successfully.")

finally:

print("Load operation completed.")