**Day-22 Evening Assessment**

11. class Library:  
   \_instance = None  # for Singleton (used later)  
  
   def \_\_init\_\_(self):  
       self.books = []  
       self.members = []  
  
   @property  
   def available\_books\_count(self):  
       return sum(1 for book in self.books if book.isAvailable)  
  
12. from abc import ABC, abstractmethod  
  
class Person(ABC):  
   def \_\_init\_\_(self, name, member\_id):  
       self.name = name  
       self.member\_id = member\_id  
  
   @abstractmethod  
   def get\_role(self):  
       pass  
  
  
class Member(Person):  
   def get\_role(self):  
       return "Member"  
  
  
class Librarian(Person):  
   def get\_role(self):  
       return "Librarian"  
  
13. 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"  
  
14. class Book:  
   def \_\_init\_\_(self, title, author, isbn):  
       self.title = title  
       self.author = author  
       self.isbn = isbn  
       self.isAvailable = True  
  
   def \_\_str\_\_(self):  
       return f"'{self.title}' by {self.author}"  
  
   def \_\_repr\_\_(self):  
       return f"Book(title={self.title}, author={self.author}, isbn={self.isbn})"  
  
15. 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. class MemberFactory:  
   @staticmethod  
   def create\_member(member\_type, name, member\_id):  
       if member\_type == "student":  
           return StudentMember(name, member\_id)  
       elif member\_type == "faculty":  
           return FacultyMember(name, member\_id)  
       elif member\_type == "research":  
           return ResearchScholar(name, member\_id)  
       else:  
           return Member(name, member\_id)  
  
17. 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  
  
   def add\_book(self, book):  
       self.books.append(book)  
       return self  # supports chaining  
  
   def register\_member(self, member):  
       self.members.append(member)  
       return self  # supports chaining

18. import json  
  
class JsonMixin:  
   def to\_json(self):  
       return json.dumps(self.\_\_dict\_\_)  
  
   @classmethod  
   def from\_json(cls, json\_str):  
       data = json.loads(json\_str)  
       return cls(\*\*data)  
  
  
class Book(JsonMixin):  
   def \_\_init\_\_(self, title, author, isbn, isAvailable=True):  
       self.title = title  
       self.author = author  
       self.isbn = isbn  
       self.isAvailable = isAvailable

Usage:  
b = Book("Python", "Guido", "123")  
json\_data = b.to\_json()  
print(json\_data)  
  
new\_book = Book.from\_json(json\_data)  
print(new\_book)  
  
19. class BookNotAvailableError(Exception):  
   def \_\_init\_\_(self, message="Book is not available for checkout"):  
       super().\_\_init\_\_(message)  
  
  
class MemberNotFoundError(Exception):  
   def \_\_init\_\_(self, message="Member not found in the system"):  
       super().\_\_init\_\_(message)  
  
20. import logging  
  
logging.basicConfig(filename="library.log", level=logging.ERROR)  
  
try:  
   with open("library\_data.json", "r") as f:  
       data = f.read()  
except FileNotFoundError as e:  
   logging.error("File not found: %s", e)  
else:  
   print("File read successfully")  
finally:  
   print("File operation attempted")