DAY-22 EVENING ASSESSMENT

11.   
class LibraryPortal:  
     
   @property  
   def available\_books\_count(self) -> int:  
       return sum(1 for b in self.books.values() if b.available)  
Usage:  
  
print("Available Books:", portal.available\_books\_count)  
  
12.   
from abc import ABC, abstractmethod  
  
class Person(ABC):  
   def \_\_init\_\_(self, person\_id: str, name: str):  
       self.person\_id = person\_id  
       self.name = name  
  
   @abstractmethod  
   def display\_info(self):  
       pass  
class BaseMember(Person):  # Now inherits Person  
     
  
13. class ResearchScholar(StudentMember, FacultyMember):  
   def display\_info(self):  
       return f"Research Scholar {self.member\_id} - {self.name}"  
  
  
14.   
class Book:  
   ...  
   def \_\_str\_\_(self):  
       return f"{self.title} by {self.author} (ISBN {self.isbn})"  
  
   def \_\_repr\_\_(self):  
       return f"Book(book\_id={self.book\_id}, title={self.title}, isbn={self.isbn})"  
print(book) → user-friendly string.  
Debugging (book) → detailed internal representation.  
  
15. class LibraryPortal:  
   \_instance = None  
  
   def \_\_new\_\_(cls, \*args, \*\*kwargs):  
       if not cls.\_instance:  
           cls.\_instance = super().\_\_new\_\_(cls)  
       return cls.\_instance  
  
16. class MemberFactory:  
   @staticmethod  
   def create\_member(mtype: str, member\_id: str, name: str):  
       if mtype.lower().startswith("stud"):  
           return StudentMember(member\_id, name)  
       elif mtype.lower().startswith("fac"):  
           return FacultyMember(member\_id, name)  
       elif mtype.lower().startswith("staff"):  
           return StaffMember(member\_id, name)  
       elif mtype.lower().startswith("scholar"):  
           return ResearchScholar(member\_id, name)  
       else:  
           return BaseMember(member\_id, name)  
  
17class LibraryPortal:  
   ...  
   def insert\_book(self, book\_id, title, author, isbn):  
       ...  
       return self  # Enables chaining  
  
   def enroll\_member(self, mtype, member\_id, name):  
       ...  
       return self  
Usage:  
portal.insert\_book("B1", "Python", "Guido", "123") \  
     .enroll\_member("Student", "S1", "Alice")  
  
18. import json  
class JsonMixin:  
   def to\_json(self):  
       return json.dumps(self.\_\_dict\_\_, indent=4)  
   @classmethod  
   def from\_json(cls, json\_str):  
       return cls(\*\*json.loads(json\_str))  
  
class Book(JsonMixin):  
   ...  
  
 Error Handling & Robustness  
  
19. class BookNotAvailableError(Exception):  
   pass  
class MemberNotFoundError(Exception):  
   pass

Usage:  
def borrow(self, member\_id, book\_id):  
   m = self.members.get(member\_id)  
   if not m:  
       raise MemberNotFoundError("Member not found.")  
   b = self.books.get(book\_id)  
   if not b or not b.available:  
       raise BookNotAvailableError("Book not available.")  
  
20. def \_load(self):  
   try:  
       if os.path.exists(BOOKS\_FILE\_NG):  
           with open(BOOKS\_FILE\_NG, "r", encoding="utf-8") as f:  
               for line in f:  
                   if line.strip():  
                       b = Book.from\_line(line)  
                       self.books[b.book\_id] = b  
   except Exception as e:  
       print("Error loading books:", e)  
   else:  
       print("Books loaded successfully.")  
   finally:  
       print("Book loading attempted.")