**Day-23 Evening Assessment**

31. import datetime  
  
HOLIDAYS = {datetime.date(2025, 1, 26), datetime.date(2025, 8, 15)}  # example holidays  
  
def calculate\_due\_date(start\_date, days=14):  
   due\_date = start\_date  
   added\_days = 0  
   while added\_days < days:  
       due\_date += datetime.timedelta(days=1)  
       if due\_date.weekday() < 5 and due\_date.date() not in HOLIDAYS:  # Mon-Fri and not holiday  
           added\_days += 1  
   return due\_date  
  
32. from collections import deque  
  
class Book:  
   def \_\_init\_\_(self, title, author, isbn):  
       self.title = title  
       self.author = author  
       self.isbn = isbn  
       self.isAvailable = True  
       self.reservation\_queue = deque()  
  
   def reserve(self, member):  
       self.reservation\_queue.append(member)  
  
   def next\_in\_queue(self):  
       return self.reservation\_queue.popleft() if self.reservation\_queue else None

33. class Member:  
   def \_\_init\_\_(self, name, member\_id):  
       self.name = name  
       self.member\_id = member\_id  
       self.borrowed\_books = {}  
       # borrowed\_books[book] = {"due\_date": date, "renewed": False}  
  
   def renew\_book(self, book):  
       if book not in self.borrowed\_books:  
           raise Exception("Book not borrowed")  
       if self.borrowed\_books[book]["renewed"]:  
           raise Exception("Renewal already used")  
       self.borrowed\_books[book]["due\_date"] = calculate\_due\_date(datetime.date.today(), 14)  
       self.borrowed\_books[book]["renewed"] = True  
  
34. from collections import Counter  
  
class Library:  
   def \_\_init\_\_(self):  
       self.borrow\_history = []  # [(book\_title, date)]  
  
   def record\_borrow(self, book):  
       self.borrow\_history.append((book.title, datetime.date.today()))  
  
   def top\_borrowed\_books(self, month, year, top\_n=5):  
       books\_in\_month = [title for (title, d) in self.borrow\_history if d.month == month and d.year == year]  
       return Counter(books\_in\_month).most\_common(top\_n)  
  
**F. Performance & Optimization**  
  
35. class Library:  
   def iter\_books(self):  
       for book in self.books:  
           yield book

Usage:  
for b in library.iter\_books():  
   print(b.title)  
  
36. import cProfile  
  
def run\_library\_simulation():  
      pass  
  
cProfile.run("run\_library\_simulation()")  
  
37. from functools import lru\_cache  
  
class Library:  
   def \_\_init\_\_(self):  
       self.books = []  
  
   @lru\_cache(maxsize=128)  
   def get\_book\_by\_isbn(self, isbn):  
       for book in self.books:  
           if book.isbn == isbn:  
               return book  
       return None  
  
38. from multiprocessing import Pool  
  
def simulate\_borrow(member\_book\_tuple):  
   member, book = member\_book\_tuple  
   try:  
       library.borrow\_book(member, book)  
   except Exception as e:  
       return str(e)  
   return f"{member.name} borrowed {book.title}"  
  
with Pool(10) as p:

   results = p.map(simulate\_borrow, [(m, b) for m, b in zip(members, books[:100])])  
  
39. from collections import defaultdict, OrderedDict  
  
borrow\_stats = defaultdict(int)  
borrow\_stats["Python101"] += 1    
ordered\_books = OrderedDict()  
ordered\_books["123"] = "Python101"  
  
40. import time, json, pickle  
  
def benchmark(n=10000):  
   books = [Book(f"Title{i}", "Author", str(i)) for i in range(n)]  
  
   start = time.time()  
   with open("books.json", "w") as f:  
       json.dump([b.\_\_dict\_\_ for b in books], f)  
   json\_time = time.time() - start  
  
   start = time.time()  
   with open("books.pkl", "wb") as f:  
       pickle.dump(books, f)  
   pickle\_time = time.time() - start  
  
   return json\_time, pickle\_time  
  
print("JSON vs Pickle:", benchmark())