import os  
from datetime import datetime, timedelta  
from typing import Dict, List  
  
DELIM = "|"  
BOOKS\_FILE\_NG = "ng\_books.txt"  
MEMBERS\_FILE\_NG = "ng\_members.txt" # includes member type  
LOAN\_DAYS\_STUDENT = 10  
LOAN\_DAYS\_FACULTY = 21  
FINE\_STUDENT = 3  
FINE\_FACULTY = 1  
  
  
class Book:  
 def \_\_init\_\_(self, book\_id: str, title: str, author: str, isbn: str, available: bool = True):  
 self.book\_id = book\_id  
 self.title = title  
 self.author = author  
 self.isbn = isbn  
 self.available = available  
  
  
 def to\_line(self):  
 return DELIM.join([self.book\_id, self.title, self.author, self.isbn, "1" if self.available else "0"])+ "\n"  
  
  
 @staticmethod  
 def from\_line(line: str) -> "Book":  
 p = line.strip().split(DELIM)  
 return Book(p[0], p[1], p[2], p[3], p[4] == "1")  
  
  
class BaseMember:  
 def \_\_init\_\_(self, member\_id: str, name: str):  
 self.member\_id = member\_id  
 self.name = name  
 self.\_\_borrowed: List[dict] = [] # encapsulated list of dicts {book\_id, issue\_date, due\_date}  
  
  
 # encapsulation with property (read-only copy)  
 @property  
 def borrowed(self) -> List[dict]:  
 return list(self.\_\_borrowed)  
  
  
 # protected-like helpers  
 def \_add\_loan(self, book\_id: str, issue\_date: datetime, due\_date: datetime):  
 self.\_\_borrowed.append({  
 "book\_id": book\_id,  
 "issue\_date": issue\_date.isoformat(),  
 "due\_date": due\_date.isoformat(),  
 })  
  
  
 def \_remove\_loan(self, book\_id: str):  
 self.\_\_borrowed = [x for x in self.\_\_borrowed if x["book\_id"] != book\_id]  
 # methods expected to be overridden  
  
 def loan\_days(self) -> int:  
 return 14  
  
 def fine\_per\_day(self) -> int: return 2  
  
 def display\_info(self) -> str:  
 return f"Member {self.member\_id} - {self.name}"  
  
  
 def to\_line(self) -> str:  
 # type|id|name|borrowed\_serialized  
 borrowed\_ser = ";".join([f"{x['book\_id']},{x['issue\_date']},{x['due\_date']}" for x in self.borrowed])  
 return DELIM.join([self.\_\_class\_\_.\_\_name\_\_, self.member\_id, self.name, borrowed\_ser]) +"\n"  
  
 @ staticmethod  
 def from\_line(line: str) -> "BaseMember":  
 p = line.strip().split(DELIM)  
 mtype, mid, name = p[0], p[1], p[2]  
 borrowed\_ser = p[3] if len(p) > 3 else ""  
 if mtype == "StudentMember":  
 m: BaseMember = StudentMember(mid, name)  
 elif mtype == "FacultyMember":  
 m = FacultyMember(mid, name)  
 else:  
 m = BaseMember(mid, name)  
 if borrowed\_ser:  
 items = borrowed\_ser.split(";")  
 for item in items:  
 if not item: continue  
 bid, issue, due = item.split(",")  
 m.\_add\_loan(bid, datetime.fromisoformat(issue), datetime.fromisoformat(due))  
 return m  
  
  
class StudentMember(BaseMember):  
 def loan\_days(self) -> int:  
 return LOAN\_DAYS\_STUDENT  
  
 def fine\_per\_day(self) -> int: return FINE\_STUDENT  
  
 def display\_info(self) -> str:  
 return f"Student {self.member\_id} - {self.name}(Loan: {self.loan\_days()} days, Fine/day:{self.fine\_per\_day()})"  
  
class FacultyMember(BaseMember):  
 def loan\_days(self) -> int:  
 return LOAN\_DAYS\_FACULTY  
  
 def fine\_per\_day(self) -> int: return FINE\_FACULTY  
  
  
 def display\_info(self) -> str:  
 return f"Faculty {self.member\_id} - {self.name} (Loan: {self.loan\_days()} days, Fine/day:{self.fine\_per\_day()})"  
  
class LibraryPortal:  
 def \_\_init\_\_(self):  
 self.books: Dict[str, Book] = {}  
 self.members: Dict[str, BaseMember] = {}  
 self.\_load()  
  
 # persistence  
 def \_load(self):  
 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  
 if os.path.exists(MEMBERS\_FILE\_NG):  
 with open(MEMBERS\_FILE\_NG, "r", encoding="utf-8") as f:  
 for line in f:  
 if line.strip():  
 m = BaseMember.from\_line(line)  
 self.members[m.member\_id] = m  
  
 def \_save(self):  
 with open(BOOKS\_FILE\_NG, "w", encoding="utf-8") as f:  
 for b in self.books.values():  
 f.write(b.to\_line())  
 with open(MEMBERS\_FILE\_NG, "w", encoding="utf-8") as f:  
 for m in self.members.values():  
 f.write(m.to\_line())  
  
 # admin  
 def insert\_book(self, book\_id: str, title: str, author: str, isbn: str):  
 if book\_id in self.books:  
 print("Book ID exists")  
 return  
 self.books[book\_id] = Book(book\_id, title, author, isbn)  
 self.\_save()  
 print("Book inserted")  
  
  
 def discard\_book(self, book\_id: str):  
 b = self.books.get(book\_id)  
 if not b:  
 print("Book not found")  
 return  
 if not b.available:  
 print("Cannot discard issued book")  
 return  
 del self.books[book\_id]  
 self.\_save()  
 print("Book discarded")  
  
 def enroll\_member(self, mtype: str, member\_id: str, name: str):  
 if member\_id in self.members:  
 print("Member ID exists")  
 return  
 if mtype.lower().startswith("stud"):  
 m = StudentMember(member\_id, name)  
 elif mtype.lower().startswith("fac"):  
 m = FacultyMember(member\_id, name)  
 else:  
 m = BaseMember(member\_id, name)  
 self.members[member\_id] = m  
 self.\_save()  
 print("Member enrolled")  
  
 # member ops  
 def borrow(self, member\_id: str, book\_id: str):  
 m = self.members.get(member\_id)  
 b = self.books.get(book\_id)  
 if not m or not b:  
 print("Invalid member/book")  
 return  
 if not b.available:  
 print("Book not available")  
 return  
 issue = datetime.now()  
 due = issue + timedelta(days=m.loan\_days())  
 # add loan via encapsulated method  
 m.\_add\_loan(book\_id, issue, due)  
 b.available = False  
 self.\_save()  
 print(f"Issued '{b.title}' to {m.name}. Due: {due.date()}")  
  
  
 def receive(self, member\_id: str, book\_id: str):  
 m = self.members.get(member\_id)  
 b = self.books.get(book\_id)  
 if not m or not b:  
 print("Invalid member/book")  
 return  
 # check active loan  
 loan = None  
 for x in m.borrowed:  
 if x["book\_id"] == book\_id:  
 loan = x  
 break  
 if not loan:  
 print("No active loan found for this book.")  
 return  
 now = datetime.now()  
 due = datetime.fromisoformat(loan["due\_date"])  
 late\_days = max(0, (now.date() - due.date()).days)  
 fine = late\_days \* self.members[member\_id].fine\_per\_day()  
  
 # update state  
 m.\_remove\_loan(book\_id)  
 b.available = True  
 self.\_save()  
 print(f"Returned '{b.title}'. Fine: {fine}")  
  
  
 # views  
 def view\_books(self):  
 for b in self.books.values():  
 print(f"[{b.book\_id}] {b.title} by {b.author} (ISBN {b.isbn}) — {'Available' if b.available else 'Issued'}")  
  
 def view\_members(self):  
 for m in self.members.values():  
 print(m.display\_info())  
 if m.borrowed:  
 for loan in m.borrowed:  
 print(f" - {loan['book\_id']} (Due {loan['due\_date']})")  
  
  
def run\_cli():  
 portal = LibraryPortal()  
 MENU = """  
NextGen Library Portal  
1)Insert Book  
2)Discard Book  
3)Enroll Member (Student/Faculty)  
4)Borrow  
5)Receive (Return)  
6)View Books  
7)View Members  
0) Exit   
Choice: """  
 while True:  
 choice = input(MENU).strip()  
 if choice == "1":  
 portal.insert\_book(input("Book ID: "), input("Title: "), input("Author: "), input("ISBN: "))  
 elif choice == "2":  
 portal.discard\_book(input("Book ID: "))  
 elif choice == "3":  
 portal.enroll\_member(input("Type (Student/Faculty): "), input("Member ID: "), input("Name: "))  
 elif choice == "4":  
 portal.borrow(input("Member ID: "), input("Book ID: "))  
 elif choice == "5":  
 portal.receive(input("Member ID: "), input("Book ID: "))  
 elif choice == "6":  
 portal.view\_books()  
 elif choice == "7":  
 portal.view\_members()  
 elif choice == "0":  
 break  
 else:  
 print("Invalid choice.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 run\_cli()  
 pass