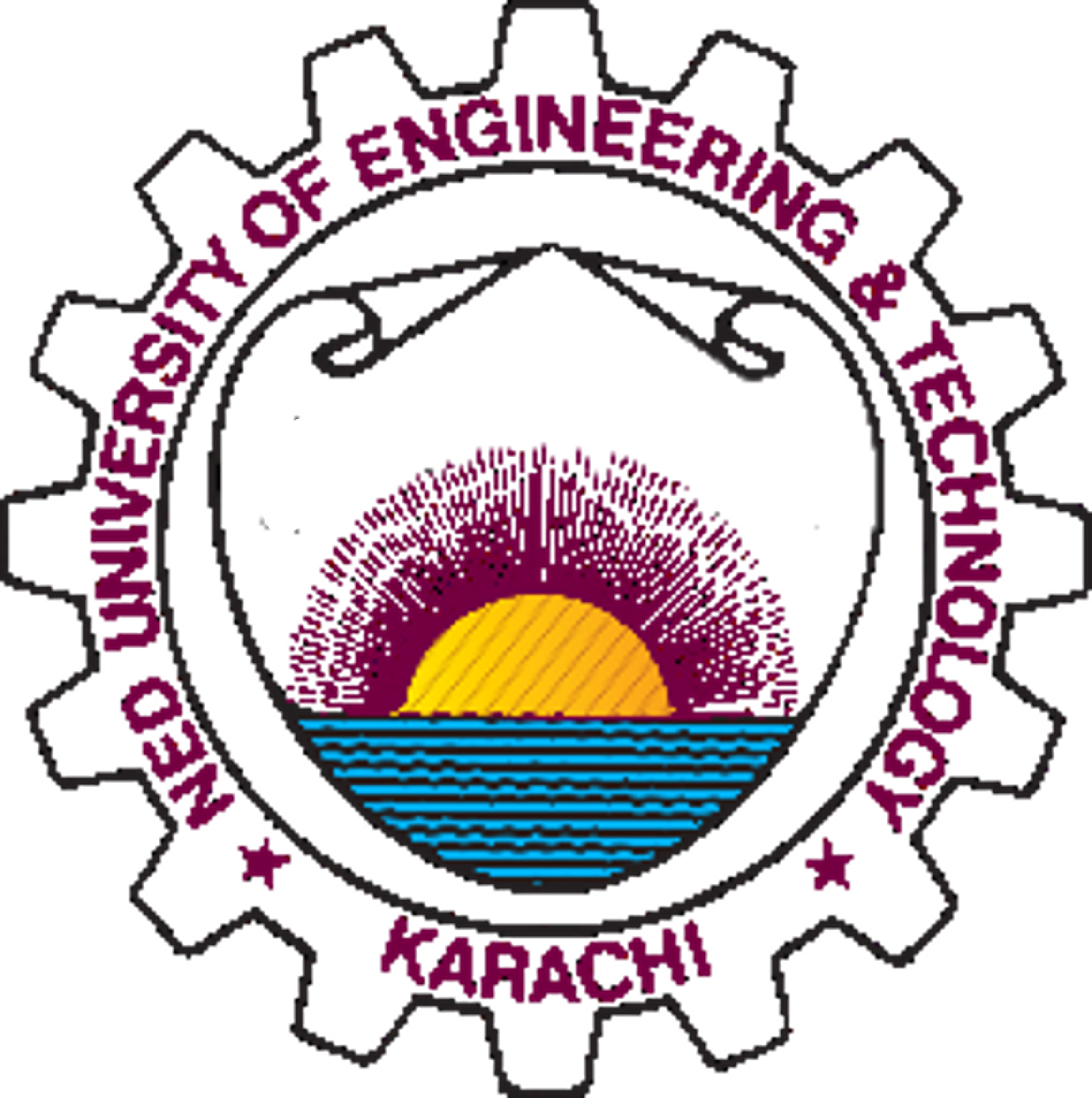
****

**NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY**

**ASSIGNMENT:**

**EXCEPTION HANDLING**

**COURSE NAME: COURSE CODE:**

OBJECT ORIENTED CS-116

PROGRAMMMING

**SUBMITTED TO:**

Ms. Ramish Fatima

**SUBMITTED BY:**

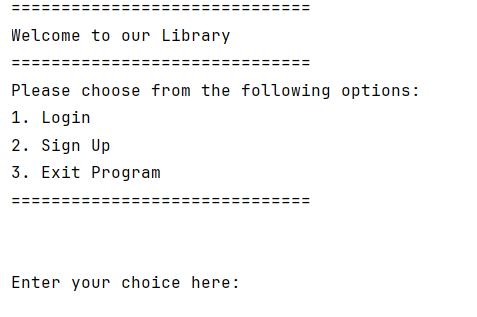
Usman Rasheed Siddiqui (CS-24038)

Huzaifa Hanif (CS-24039)

**CODE:**

*# Custom exception classes for handling specific exceptions in the Library System*class UserNotFoundError(Exception):  
 *"""Custom exception for when a user is not found in the system"""* def \_\_init\_\_(self):  
 self.message = 'User not found'  
 super().\_\_init\_\_(self.message)  
  
 def \_\_str\_\_(self):  
 return self.message  
  
class PasswordNotMatchError(Exception):  
 *"""Custom exception for when password doesn't match user records"""* def \_\_init\_\_(self):  
 self.message = 'Password does not match'  
 super().\_\_init\_\_(self.message)  
  
 def \_\_str\_\_(self):  
 return self.message  
  
class AlreadyExistError(Exception):  
 *"""Custom exception for duplicate usernames/passwords"""* pass *# Raised when a username or password already exist*class BookNotAvailableError(Exception):  
 *"""Custom exception when trying to borrow unavailable book"""* pass *# Raised when a book is not available for borrowing*class BookNotFoundError(Exception):  
 *"""Custom exception when book doesn't exist in library"""* pass *# Raised when a book doesn't exist in the library*class BalanceError(Exception):  
 *"""Custom exception for invalid balance operations"""* pass *# Raised for balance-related issues*class PasswordRangeError(Exception):  
 *"""Custom exception for password length violations"""* pass *# Raised when password doesn't meet length requirements*class BookNotBorrowedError(Exception):  
 *"""Custom exception when returning unborrowed book"""* pass *# Raised when trying to return a book that wasn't borrowed*class Library:  
 def \_\_init\_\_(self, name="", password="", book=""):  
 *# Initialize library with default values* self.name = name *# Current user's name* self.password = password *# Current user's password* self.book = book *# Current book being processed* self.cost = 0 *# Cost of current book* self.balance = 0 *# Current user's balance* self.main\_menu() *# Start with main menu  
  
 # Predefined user accounts with passwords, borrowed books, and balances* \_accounts = {  
 "Alice": {  
 "password": "alice123",  
 "borrowed": [],  
 "balance": 500,  
 },  
 "Ahmed": {  
 "password": "Ahmed#886",  
 "borrowed": [],  
 "balance": 300,  
 },  
 "Charlie": {  
 "password": "charliePwd!",  
 "borrowed": [],  
 "balance": 750,  
 },  
 "Ayesha": {  
 "password": "AyeshaSecret",  
 "borrowed": [],  
 "balance": 1000,  
 },  
 "Ali": {  
 "password": "Ali\_e",  
 "borrowed": [],  
 "balance": 200,  
 }  
 }  
  
 *# Predefined books with their details* \_books = {  
 "101": {"title": "Dr. Jekyll and Mr. Hyde", "cost": 100, "available": True},  
 "102": {"title": "To Kill a Mockingbird", "cost": 250, "available": True},  
 "103": {"title": "1984", "cost": 180, "available": True},  
 "104": {"title": "The Great Gatsby", "cost": 220, "available": False},  
 "105": {"title": "Pride and Prejudice", "cost": 150, "available": True},  
 "106": {"title": "The Hobbit", "cost": 300, "available": True},  
 "107": {"title": "Harry Potter and the Sorcerer's Stone", "cost": 350, "available": False},  
 "108": {"title": "The Catcher in the Rye", "cost": 190, "available": True},  
 "109": {"title": "Brave New World", "cost": 210, "available": True},  
 "110": {"title": "The Alchemist", "cost": 170, "available": True}  
 }  
  
 @classmethod  
 def accounts(cls):  
 *"""Class method to access accounts dictionary"""* return cls.\_accounts  
  
 @classmethod  
 def books(cls):  
 *"Class method to access books dictionary"* return cls.\_books  
  
 def quit\_choice(self, choice):  
 *"""Check if user wants to quit (entered 'q' or 'Q')"""* if choice == "q" or choice == "Q":  
 return True  
 return False  
  
 def login(self):  
 *"""Handle user login process"""* print()  
 print("=" \* 30)  
 print("Login")  
 print("=" \* 30)  
 print()  
 print("Enter q/Q at any time to quit to main menu.")  
  
 *# Get username and password* self.name = input("Enter your name: ").strip()  
 if self.quit\_choice(self.name):  
 return self.main\_menu()  
 self.password = input("Enter your password: ").strip()  
 if self.quit\_choice(self.password):  
 return self.main\_menu()  
  
 *# Validate credentials* while True:  
 try:  
 *# Get username and password with quit option* user\_found = False  
 for account, details in Library.accounts().items():  
 if account.lower() == self.name.lower():  
 user\_found = True  
 if details["password"] == self.password:  
 self.name = account  
 print(f"Password matched\nWelcome Mr./Mrs. {self.name.upper()}")  
 return True *# Successful login* else:  
 *# EXCEPTION: Wrong password* raise PasswordNotMatchError  
  
 *# EXCEPTION: User not found* if not user\_found:  
 raise UserNotFoundError()  
  
 except UserNotFoundError as e: *# HANDLE: User doesn't exist* print("Error:",e)  
 break  
 except PasswordNotMatchError as e: *# HANDLE: Incorrect password* print("Error:",e)  
 break  
  
 def make\_account(self):  
 *"""Handle new account creation"""* print()  
 print("="\*30)  
 print("Sign Up")  
 print("="\*30)  
 print()  
 print("Enter q/Q at any time to quit to main menu.")  
  
 *# Get and validate username* while True:  
 try:  
 self.name = input("Enter your name: ").strip()  
 if self.quit\_choice(self.name):  
 return self.main\_menu()  
  
 *# EXCEPTION: Invalid name format* if not self.name.isalpha() or not self.name:  
 raise TypeError("Enter Correct Name (letter only, no spaces allowed)")  
 for account in Library.accounts().keys():  
  
 *# EXCEPTION: Username already exists* if account.lower() == self.name.lower():  
 raise AlreadyExistError()  
 break  
 except TypeError as e: *# HANDLE: Invalid name format* print("Error:",e)  
 except AlreadyExistError: *# HANDLE: Duplicate username* print("Username already exists")  
  
 *# Get and validate password* while True:  
 try:  
 self.password = input("Enter your password (8+ characters): ").strip()  
 if self.quit\_choice(self.password):  
 return self.main\_menu()  
  
 *# EXCEPTION: Empty password* if not self.password:  
 raise ValueError("This field is required")  
  
 *# EXCEPTION: Password too short* if len(self.password) < 8:  
 raise PasswordRangeError("Password must be at least 8 characters long")  
 for account in self.accounts().values():  
  
 *# EXCEPTION: Duplicate password* if account["password"] == self.password:  
 raise AlreadyExistError  
 break  
 except ValueError as e: *# HANDLE: Empty password* print("Error:",e)  
 except AlreadyExistError: *# HANDLE: Duplicate password* print("Password already exists")  
 except PasswordRangeError as e: *# HANDLE: Short password* print("Error:",e)  
  
 *# Get and validate initial balance* while True:  
 try:  
 self.balance = input("Enter your balance: ")  
 if self.quit\_choice(self.balance):  
 return self.main\_menu()  
  
 *# EXCEPTION: Empty balance* if not self.balance.isdigit():  
 raise TypeError  
 self.balance = int(self.balance)  
  
 *# EXCEPTION: Too small balance* if self.balance <= 100:  
 raise ValueError("Balance must be greater than 100")  
  
 *# EXCEPTION: Too large balance* if self.balance > 5000:  
 raise OverflowError("Balance is exceeding limit. Your total balance should be at most 5000.")  
 break  
 except TypeError: *# HANDLE: Empty balance* print("Invalid balance")  
 except ValueError as e: *# HANDLE: Too small balance* print("Error:",e)  
 except OverflowError as e: *# HANDLE: Too large balance* print("Error:",e)  
  
 *# Create new account with validated details* account ={  
 "password": self.password,  
 "borrowed": [],  
 "balance": self.balance,  
 }  
  
 Library.\_accounts[self.name] = account  
  
 print("Account created successfully. Please login to access your account.")  
 return self.main\_menu()  
  
  
 def save\_book\_id(self, id):  
 *"""Update user account after borrowing a book"""* for account, details in self.accounts().items():  
 if account == self.name:  
 if details["balance"] < self.cost:  
 print("You have insufficient balance")  
 self.user\_menu()  
 break  
 details["borrowed"].append(id) *# Add book to borrowed list* details["balance"] -= self.cost *# Deduct cost from balance* print(f"You succesfully borrowed this book. Your account balance is now {details['balance']}")  
 break  
  
 def borrow\_book(self):  
 *"""Handle book borrowing process"""* print()  
 print("=" \* 30)  
 print("Borrowing Book")  
 print("=" \* 30)  
 print()  
 print("Enter q/Q at any time to quit to user menu.")  
  
 *# Taking input for book and validating the input* try:  
 self.book = input("Enter available book's id to borrow: ")  
 if self.quit\_choice(self.book):  
 return self.user\_menu()  
 book\_found = False  
  
 *# EXCEPTION: Book doesn't exist* if self.book not in Library.books():  
 raise KeyError("Book ID does not exist")  
  
 for account, details in Library.accounts().items():  
 if account == self.name:  
 *# EXCEPTION: Book already borrowed by user* if self.book in details["borrowed"]:  
 raise AlreadyExistError  
  
 *# Process book borrowing* for id, book in Library.books().items():  
 if self.book == id:  
 book\_found = True  
  
 *# EXCEPTION: Book not available* if book["available"]:  
 self.cost = book["cost"]  
 self.save\_book\_id(self.book)  
 else:  
 raise BookNotAvailableError  
  
 *# EXCEPTION: Book not found* if not book\_found:  
 raise BookNotFoundError  
 except BookNotAvailableError: *# HANDLE: Book not available* print("This book is currently not available")  
 except BookNotFoundError: *# HANDLE: Book doesn't exist* print("Sorry! We don't have this book")  
 except KeyError as e: *# HANDLE: Invalid book ID format* print("Error:",e)  
 except AlreadyExistError: *# HANDLE: Already borrowed by user* print("Book is already borrowed by you. Please choose one you did not borrow.")  
  
 return self.user\_menu()  
  
 def return\_book(self):  
 print()  
 print("=" \* 30)  
 print("Returning Book")  
 print("=" \* 30)  
 print()  
 print("Enter q/Q at any time to quit to user menu.")  
 try:  
 book\_returned = False  
 for account, details in Library.accounts().items():  
 if account == self.name:  
  
 *# EXCEPTION: No borrowed books* if not details["borrowed"]:  
 raise BookNotBorrowedError("No books borrowed")  
 id = input("Enter book ID: ")  
 if self.quit\_choice(id):  
 return self.user\_menu()  
  
 *# EXCEPTION: Invalid book ID format* if not id.isdigit():  
 raise ValueError  
  
 *# Process return* for book\_id in details["borrowed"]:  
 if book\_id == id:  
 book\_returned = True  
 details["borrowed"].remove(id)  
 print("Book returned succesfully")  
 if id in Library.books():  
 Library.books()[id]["available"] = True  
 return self.user\_menu()  
 if not book\_returned:  
 raise BookNotBorrowedError("This book was not borrowed by you")  
  
 except ValueError: *# HANDLE: Invalid book ID format* print("Invalid book ID")  
  
 except BookNotBorrowedError as e: *# HANDLE: Book not borrowed or no books* print("Error:",e)  
  
 return self.user\_menu()  
  
 def update\_balance(self):  
 *"""Handle balance update process"""* try:  
 *# Get current user's account* for account, details in Library.accounts().items():  
 if account == self.name:  
 print("Enter q/Q at any time to quit to user menu.")  
  
 *# Get user input for balance update* update\_balance = input("Enter your new balance to be added: ")  
 if self.quit\_choice(update\_balance):  
 return self.user\_menu()  
  
 *# EXCEPTION: Validate input is numeric* if not update\_balance.isdigit():  
 raise ValueError("Please enter a appropriate balance")  
 update\_balance = int(update\_balance)  
  
 *# EXCEPTION: Check for positive amount* if update\_balance <= 0:  
 raise ValueError("Balance must be greater than 0")  
  
 *# EXCEPTION: Check total balance won't exceed 5000 limit* if (details["balance"] + update\_balance) > 5000:  
 raise OverflowError("Balance is exceeding limit. Your total balance should be at most 5000.")  
  
 *# Update the balance* details["balance"] += update\_balance  
  
 *# EXCEPTION: Low Balance* if details["balance"] < 100:  
 raise BalanceError("Balance must be greater than 100")  
 print("Balance updated successfully")  
  
 except BalanceError as e: *# HANDLE: Balance problems* print("Error:",e)  
 except ValueError as e: *# HANDLE: Invalid numeric input* print("Error:",e)  
 except OverflowError as e: *# HANDLE: Balance would exceed maximum* print("Error:", e)  
  
 def status(self):  
 *"""Display user's current status (borrowed books and balance)"""* for account, details in Library.accounts().items():  
 if account == self.name:  
 print("\n","="\*30)  
 print(f"Borrowed Books ID: {details['borrowed'] if details['borrowed'] else 'No books borrowed'}")  
 print(f"Balance: {details['balance']}")  
 print("=" \* 30)  
 print()  
  
 def view\_all\_books\_available(self):  
 *"""Display all available books"""* print("\n","="\*30)  
 print("Available Books")  
 print("="\*30)  
 for id, book in Library.books().items():  
 if book.get("available"):  
 print("ID:", id, end=" | ")  
 print(f"Title: {book['title']} | Cost: {book['cost']} | Availability: Available")  
 print("=" \* 30)  
 print()  
  
 def choice\_main(self):  
 *"""Handle main menu choices"""* while True:  
 try:  
 choice = input("\nEnter your choice here: ")  
 if choice == "1":  
 if self.login():  
 self.user\_menu()  
 else:  
 self.main\_menu()  
 elif choice == "2":  
 self.make\_account()  
 elif choice == "3":  
 print("Thank you for using our application")  
 exit()  
  
 *# EXCEPTION: Invalid entry* if not choice.isdigit():  
 raise TypeError  
 *# EXCEPTION: Invalid number entry* if int(choice) < 1 or int(choice) > 3:  
 raise ValueError  
  
 *# HANDLING: Invalid entry* except TypeError:  
 print("Error: You did not enter a number. Please enter a number from 1 ----> 3")  
  
 *# HANDLING: Invalid number entry* except ValueError:  
 print("Please enter a number from 1 ----> 3")  
  
 def choice\_user(self):  
 *"""Handle user menu choices"""* while True:  
 try:  
 print("Press q/Q to quit to main menu. (You have to then login again)")  
 choice = input("\nEnter your choice here: ")  
 if self.quit\_choice(choice):  
 return self.main\_menu()  
 if choice == "1":  
 self.borrow\_book()  
 elif choice == "2":  
 self.return\_book()  
 elif choice == "3":  
 self.update\_balance()  
 elif choice == "4":  
 self.status()  
 elif choice == "5":  
 self.view\_all\_books\_available()  
 elif choice == "6":  
 self.main\_menu()  
  
 *# EXCEPTION: Invalid entry* if not choice.isdigit():  
 raise TypeError  
 *# EXCEPTION: Invalid number entry* if int(choice) < 1 or int(choice) > 6:  
 raise ValueError  
  
 *# HANDLING: Invalid entry* except TypeError:  
 print("Error: You did not enter a number. Please enter a number from 1 ----> 6")  
  
 *# HANDLING: Invalid number entry* except ValueError:  
 print("Please enter a number from 1 ----> 6")  
  
  
 def main\_menu\_interface(self):  
 *"""Display main menu options"""* print("Please choose from the following options:")  
 print("1. Login")  
 print("2. Sign Up")  
 print("3. Exit Program")  
 print("=" \* 30)  
 print()  
  
 def main\_menu(self):  
 *"""Display main menu and handle choices"""* print("="\*30)  
 print("Welcome to our Library")  
 print("="\*30)  
 self.main\_menu\_interface()  
 self.choice\_main()  
  
 def user\_menu\_interface(self):  
 *"""Display user menu options"""* print("=" \* 30)  
 print(f"Mr./ Mrs {self.name.upper()}'s Dashboard")  
 print("=" \* 30)  
 print("Please choose from the following options:")  
 print("1. Borrow Book")  
 print("2. Return Book")  
 print("3. Update Balance")  
 print("4. Check Status")  
 print("5. View All Books Available")  
 print("6. Exit to Main Menu")  
 print("=" \* 30)  
 print()  
  
 def user\_menu(self):  
 *"""Display user menu and handle choices"""* self.user\_menu\_interface()  
 self.choice\_user()  
  
*# Create Library instance to start the program*L1 = Library()

**SNIPPETS:**

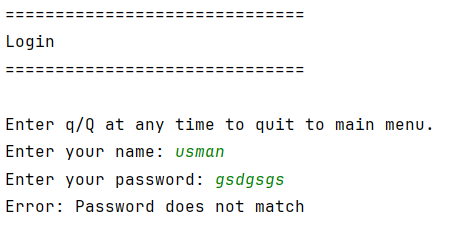
****

**A close up of a sign

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.



A screenshot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.