**Lab Exercise 9**

1. **Define a class to represent a bank account. Include the following members:**

Data Members:  
a.Name of the Depositor  
b.Account Number  
c.Type of Account  
d.Balance amount in the account

Data Methods:  
a.To assign initial values  
b.To deposit an amount  
c.To withdraw an amount  
d.To display name and balance.

**Output:**

1. **New customer**
2. **Deposit**
3. **Withdrawal**
4. **Display**
5. **Exit**

class BankAccount:

def customerMethods(self):

self.Customer={}

while True:

choice=input("Do you want to enter customer details (Y/N) : ").upper()

if(choice=='Y'):

print('''

1. New customer

2. Deposit

3. Withdrawal

4. Display

5. Exit

''')

choice=int(input("Enter choice: "))

match choice:

case 1:

self.AccountNumber= input("New Customer: Enter assign account number: ")

self.Name= input("New Customer: Enter Customer name: ")

self.TypeofAccount= input("New Customer: Enter account type such as SB or CA: ")

self.Balance=0

self.Customer[self.AccountNumber]=[self.Name,self.TypeofAccount,self.Balance]

case 2:

AccountNumber=input("Enter account number: ")

if(AccountNumber in self.Customer.keys()):

d= float(input("Enter deposit amount: "))

self.Customer[self.AccountNumber][2]=d

else:

print("Customer account number is invalid.")

case 3:

AccountNumber=input("Enter account number: ")

if(AccountNumber in self.Customer.keys()):

self.Withdrawal= float(input("Enter withdraw amount: "))

if (self.Withdrawal<=self.Customer[self.AccountNumber][2]):

self.Customer[self.AccountNumber][2]=self.Customer[self.AccountNumber][2]-self.Withdrawal

print("Balance is : ",self.Customer[self.AccountNumber][2])

else:

print("You can not withdraw more than balance.")

else:

print("Customer account number is invalid.")

case 4:

print("Customer accounts: in the order of Account number,name,account type,balance",self.Customer)

case 5:

print("Exited.")

else:

break

print("Customer accounts: in the order of Account number,name,account type,balance",self.Customer)

a1=BankAccount()

a1.customerMethods()

Do you want to enter customer details (Y/N) : y

1. New customer

2. Deposit

3. Withdrawal

4. Display

5. Exit

Enter choice: 1

New Customer: Enter assign account number: 1

New Customer: Enter Customer name: o

New Customer: Enter account type such as SB or CA: s

Do you want to enter customer details (Y/N) : y

1. New customer

2. Deposit

3. Withdrawal

4. Display

5. Exit

Enter choice: 2

Enter account number: 1

Enter deposit amount: 10

Do you want to enter customer details (Y/N) : y

1. New customer

2. Deposit

3. Withdrawal

4. Display

5. Exit

Enter choice: 3

Enter account number: 1

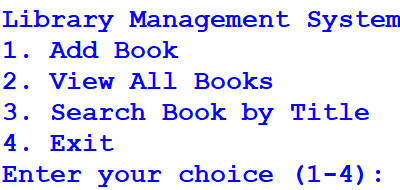
Enter deposit amount: 5

Balance is : 5.0

Do you want to enter customer details (Y/N) : n

Customer accounts: in the order of Account number,name,account type,balance {'1': ['o', 's', 5.0]}

1. **Create a menu-driven program that allows users to interact with different functionalities related to classes and objects. The example should include a simple system to manage a library, where you can add books, view books, and search for books. Illustrate the use of classes, objects, and user-driven menus as shown below.**



**Classes and Methods will be as follows:**

**Book Class:** should Represents a book with title, author, and year attributes. The \_\_str\_\_ method provides a string representation for easy printing.

**Library Class**: Manages a list of Book objects. It includes methods to add a book (add\_book), view all books (view\_books), and search for a book by title (search\_book).

**Menu Functionality will be as follows:**

**print\_menu** Function: Prints the menu options for user interaction.

**main Function:** Runs the menu-driven program loop. It prompts the user to make a choice and performs the corresponding action (add, view, search, or exit).

**User Interaction will be as follows:**

The program interacts with the user through the console. It handles book additions, views, and searches based on user input

class BookDetails:

''' making the BookDetails class with objects to send as a input to Library class as an object'''

def \_\_init\_\_(self,title,author,year):

self.title=title

self.author=author

self.year=year

'''string representation of an object for easy printing

The \_\_str\_\_() dunder method returns a reader-friendly string representation of a class object.

It can be called with the built-in str() and print() functions'''

def \_\_str\_\_(self):

return f" '{self.title}': author is {self.author},{self.year}."

class Library:

def \_\_init\_\_(self):

self.libraryBooks=[] #This list object is used to prepare library books list

#method to add new book to library

def addBooksToLibraryBooksList(self,b):

# Here b is the object of the BookDetails class which is passed as an argument here and added to library list

self.libraryBooks.append(b)

print(f" Book '{b.title}' is added to library.")

#method to view all books in library

def methodToViewBooks(self):

if not self.libraryBooks:

print("Library is empty.")

else:

print("Books in library: ")

for index,bookName in enumerate(self.libraryBooks,1):

print(f"{index}.{bookName}")

#method to search for a book by title

def search\_book(self,t):

book\_found=[book for book in self.libraryBooks if t.lower() in book.title.lower()]

if book\_found:

print("Book is found.")

for book in book\_found:

print(book)

else:

print(f"Not found with title '{t}'")

# menu options function

def menu():

print('''Library Menu:

1. Add a Book

2. View All Books

3. Search for a Book by Title

4. Exit

''')

# main function to run the menu driven program

def main():

library=Library() #library object is invoked

while True:

menu()

choice=int(input("Enter choice: "))

match choice:

case 1:

ti=input("Enter title of the book: ")

au=input("Enter author of the book: ")

y=input("Enter year of the book: ")

new\_book=BookDetails(ti,au,y) #object of the BookDetails class is instantiated

library.addBooksToLibraryBooksList(new\_book) #object of the BookDetails class i.e., new\_book is passed an argument to library method

case 2:

library.methodToViewBooks()

case 3:

titleOfBook=input("Enter title of the book to be searched.")

library.search\_book(titleOfBook)

case 4:

print("Exiting the program.")

break

case \_\_:

print("Invalid choice.")

#to start the program

if \_\_name\_\_=="\_\_main\_\_":

main()

**OutPut:**

Library Menu:

1. Add a Book

2. View All Books

3. Search for a Book by Title

4. Exit

Enter choice: 1

Enter title of the book: o

Enter author of the book: m

Enter year of the book: 2024

Book 'o' is added to library.

Library Menu:

1. Add a Book

2. View All Books

3. Search for a Book by Title

4. Exit

Enter choice: 1

Enter title of the book: a

Enter author of the book: b

Enter year of the book: 2023

Book 'a' is added to library.

Library Menu:

1. Add a Book

2. View All Books

3. Search for a Book by Title

4. Exit

Enter choice: 2

Books in library:

1. 'o': author is m,2024.

2. 'a': author is b,2023.

Library Menu:

1. Add a Book

2. View All Books

3. Search for a Book by Title

4. Exit

Enter choice: 3

Enter title of the book to be searched. a

Book is found.

'a': author is b,2023.

Library Menu:

1. Add a Book

2. View All Books

3. Search for a Book by Title

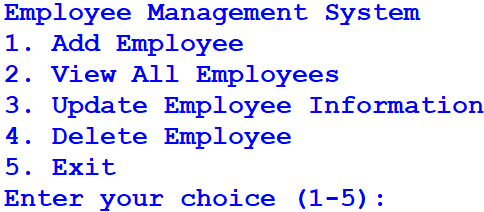
4. Exit

Enter choice: 4

Exiting the program.

1. **Create a menu-driven Employee Management System using Python classes and objects. The system will allow the user to:**
2. Add new employees with details.
3. View all employees.
4. Update employee information.
5. Delete an employee record.
6. Exit the program.

**Output should display a menu like this:**

****

**class Employee:**

**def \_\_init\_\_(self):**

**self.emp={}**

**def addEmployee(self):**

**while True:**

**choice=int(input("1 to add employee with details And 0 to exit: "))**

**match choice:**

**case 1:**

**name=input("Enter name of employee: ")**

**details=input("Enter details of employee: ")**

**self.name=name**

**self.details=details**

**self.emp[self.name]=self.details**

**case \_\_:**

**print("exit")**

**break**

**def viewEmployees(self):**

**print("employees: ",self.emp)**

**def updateEmployee(self):**

**while True:**

**choice=int(input("1 to update employee details And 0 to exit: "))**

**match choice:**

**case 1:**

**name=input("Enter name of employee: ")**

**if name in self.emp:**

**details=input("Enter details of employee: ")**

**self.emp[name]=details**

**else:**

**print("Employee is not found.")**

**break**

**case \_\_:**

**print("exit")**

**break**

**def deleteEmployee(self):**

**while True:**

**choice=int(input("1 to delete employee with details And 0 to exit: "))**

**match choice:**

**case 1:**

**name=input("Enter name of employee: ")**

**if name in self.emp:**

**del(self.emp[name])**

**else:**

**print("Employee is not found.")**

**break**

**case \_\_:**

**print("exit")**

**break**

**def choiceEmployee(self):**

**choice=int(input('''**

**1. Add new employees with details.**

**2. View all employees.**

**3. Update employee information.**

**4. Delete an employee record.**

**5. Exit the program. :'''))**

**match choice:**

**case 1:**

**self.addEmployee()**

**self.viewEmployees()**

**case 2:**

**self.viewEmployees()**

**case 3:**

**self.updateEmployee()**

**print("Employees after update:")**

**self.viewEmployees()**

**case 4:**

**self.deleteEmployee()**

**print("Employees after delete:")**

**self.viewEmployees()**

**case 5:**

**print("exited")**

**case \_\_:**

**print("Invalid choice.:")**

**e1=Employee()**

**e1.choiceEmployee()**

4. Write a program to implement following.

Create a base class named person consisting of name and code. Create 2 child classes

1. **Account** with member\_pay
2. **Admin** with experience and inherit the base class.

Create a class **employee** with name, code, experience and pay by inheriting the above classes.

class Person:

def \_\_init\_\_(self,name,code):

name=input("enter name: ")

code=int(input("Enter code : "))

self.name=name

self.code=code

class Account(Person):

def \_\_init\_\_(self,pay):

print("Account class is invoked through Employee child class.")

pay=float(input("Enter pay: "))

self.pay=pay

print("Person class is invoked through Admin child class.")

Person.\_\_init\_\_(self,'x','y')

class Admin(Person):

def \_\_init\_\_(self,experience):

print("Admin class is invoked through Employee child class.")

experience=float(input("Enter experience : "))

self.experience=experience

class Employee(Account,Admin):

def \_\_init\_\_(self):

Account.\_\_init\_\_(self,'x')

Admin.\_\_init\_\_(self,'y')

**e1=Employee()**

**print(f"name is:{e1.name}; code is {e1.code}; experience is {e1.experience}; pay is {e1.pay}.")**

Account class is invoked through Employee child class.

Enter pay: 20000

Person class is invoked through Admin child class.

enter name: om

Enter code : 2

Admin class is invoked through Employee child class.

Enter experience : 5

name is:om; code is 2; experience is 5.0; pay is 20000.0.

**5.** Write program to create a base class staff with code and name. Derive classes teacher(subject , publication) , typist (speed) , officer (grade) . Using the typist class as base class,create two classes regular(salary) and casual(daily wages).Implement a menu driven program for the same.

**Output:**

1. **Teacher**
2. **Officer,**
3. **Regular Typist**
4. **Casaul typist**
5. **Exit**

**class staff:**

**def \_\_init\_\_(self,code,name):**

**self.code=code**

**self.name=name**

**class teacher(staff):**

**def \_\_init\_\_(self,subject,publication):**

**self.subject=subject**

**self.publication=publication**

**def \_\_str\_\_(self):**

**return f"Teacher details\n: Code is {self.code}\nname is {self.name}\nsubject is {self.subject}\npublications is {self.publication}."**

**class typist(staff):**

**def \_\_init\_\_(self,speed):**

**self.speed=speed**

**class officer(staff):**

**def \_\_init\_\_(self,grade):**

**self.grade=grade**

**class regular(typist):**

**def \_\_init\_\_(self,salary):**

**self.salary=salary**

**class casual(typist):**

**def \_\_init\_\_(self,dailyWages):**

**self.dailyWages=dailyWages**

**def menu():**

**print('''**

**1.Teacher**

**2.Officer,**

**3.Regular Typist**

**4.Casaul typist**

**5.Exit**

**''')**

**def main():**

**while True:**

**menu()**

**choice=int(input("Enter Choice: "))**

**match choice:**

**case 1:**

**subject=input("Enter subject: ")**

**publication=input("Enter publications: ")**

**code=input("Enter code: ")**

**name=input("Enter name: ")**

**t=teacher(subject,publication)**

**t.code=code**

**t.name=name**

**print(f"\nTeacher details:\nCode is {t.code}\nname is {t.name}\nsubject is {t.subject}\npublications is {t.publication}.")**

**case 2:**

**grade=input("Enter grade: ")**

**code=input("Enter code: ")**

**name=input("Enter name: ")**

**o=officer(grade)**

**o.code=code**

**o.name=name**

**print(f"\nofficer details:\nCode is {o.code}\nname is {o.name}\ngrade is {o.grade}.")**

**case 3:**

**code=input("Enter code: ")**

**name=input("Enter name: ")**

**speed=float(input("enter speed: "))**

**rt=regular(speed)**

**rt.speed=int(input("enter speed: "))**

**rt.code=code**

**rt.name=name**

**print(f"\nregularTypist details:\nCode is {rt.code}\nname is {rt.name}\nspeed is {rt.speed}\nsalary is {rt.salary}.")**

**case 4:**

**code=input("Enter code: ")**

**name=input("Enter name: ")**

**dailyWages=float(input("enter dailyWages: "))**

**ct=casual(dailyWages)**

**ct.speed=int(input("enter speed: "))**

**ct.code=code**

**ct.name=name**

**print(f"\ncasualTypist details:\nCode is {ct.code}\nname is {ct.name}\nspeed is {ct.speed}\ndailyWages is {ct.dailyWages}.")**

**case 5:**

**print("Exiting")**

**break**

**case \_\_:**

**break**

**main()**

**o/p:**

1.Teacher

2.Officer,

3.Regular Typist

4.Casaul typist

5.Exit

Enter Choice: 1

Enter subject: phy

Enter publications: 5

Enter code: 25

Enter name: om

Teacher details:

Code is 25

name is om

subject is phy

publications is 5.

1.Teacher

2.Officer,

3.Regular Typist

4.Casaul typist

5.Exit