

CONCEPTS COVERED

- ABSTRACTION
- ENCAPSULATION
- GETTER AND SETTER
- INHERITANCE

1. Library Management System

Scenario: You are tasked with developing a library management system where users can borrow and return books.

Requirements:

Create a Book class that has the following private attributes: title, author, and isbn. Implement methods to get and set the values of these attributes (getters and setters). Create a Library class that has a list of Book objects. Implement methods to add a new book, remove a book by its ISBN, and list all available books. Use encapsulation to ensure that the books can only be manipulated through the Library class. Task:

Implement the Book and Library classes with proper encapsulation. Add methods to add, remove, and list books in the library

```
class book:
    def __init__(self,title,author,isbn):
        self.__title = title
        self.__author = author
        self.__isbn = isbn
    def author(self):
        return self.__author
    def title(self):
        return self.__title
    def isbn(self):
        return self.__isbn
class library:
    def __init__(self):
        self.lis = []
    def add(self,b):
        self.lis.append(b)
    def remove(self,b,isbn):
        self.lis.remove(b)
        for book in self.lis:
            if book.isbn()==isbn:
                self.lis.remove(book)
                print(f"Book with ISBN {isbn} removed.")
                return
        print(f"Book with ISBN {isbn} not found in the library.")

    def display(self):
        for x in self.lis:
            print("BOOK:")
            print("Author: ",x.author())
            print("Title: ",x.title())
            print("ISBN: ",x.isbn(),"\n")
b1 = book("A","B",123)
b2 = book("c","D",456)
l = library()
l.add(b1)
l.add(b2)
l.display()
l.remove(b1,456)
l.display()
```



```
BOOK:
Author:  B
Title:  A
ISBN:  123
```

```
BOOK:
Author:  D
Title:  c
ISBN:  456
```

```
Book with ISBN 456 removed.
```

2. Bank Account Management

Scenario: Develop a bank account management system where users can create accounts, deposit, and withdraw money.

Requirements:

Create a BankAccount class with private attributes: account_number, account_holder, and balance. Implement methods to get the account details and balance (getters). Implement methods to deposit and withdraw money, ensuring that the balance cannot go negative. Use encapsulation to ensure that the balance cannot be directly modified. Task:

Implement the BankAccount class with proper encapsulation. Add methods to deposit and withdraw money, and to get account details.

```
class BankAccount:
    def __init__(self,account_number,account_holder,money):
        self.__account_number=account_number
        self.__account_holder=account_holder
        self.__money=money

    def set_attribute(self,ac_no,balan):
        if ac_no%100000==0:
            self.__an = ac_no
        else:
            print("Wrong Account number!!")
        if balan >100:
            self.__bal = balan
        else:
            print("TO start the account please deposit atleast 100$")

    def get_detail(self):
        print("Account No:", {self.__account_number})
        print("Account Holder:",{self.__account_holder})
        print("Money:",{self.__money})

    def deposit(self,money):
        if money>0:
            self.__money += money
            print(money)

    def withdraw(self,money):
        if money>0:
            self.__money-=money
            print(money)
b1=BankAccount(34567,"adbn",80000)
b1.get_detail()
b1.deposit(8000)
b1.get_detail()
b1.withdraw(4000)
b1.get_detail()
```

```
→ Account No: {34567}
Account Holder: {'adbn'}
Money: {80000}
8000
Account No: {34567}
Account Holder: {'adbn'}
Money: {88000}
4000
Account No: {34567}
Account Holder: {'adbn'}
Money: {84000}
```

3. Employee Management System

- Create an Employee class with private attributes employee_id, name, position, and salary.
- Implement getter and setter methods for position.
- Implement a setter method for salary that ensures the salary being set is positive.

```

class Employee:
    def __init__(self,employee_id,name,position,salary):
        self.__employee_id=employee_id
        self.__name=name
        self.__position=position
        self.__salary=salary
    def set_attribute(self,slry):
        if slry>0:
            self.__salary=slry
        else:
            print("no salary is there")
    def emp_id(self):
        return self.__employee_id
    def name(self):
        return self.__name
    def pos(self):
        return self.__position
    def sal(self):
        return self.__salary
class department:
    def __init__(self):
        self.pos = []
    def add(self,a):
        self.pos.append(a)
        print("New Employee is added!!")
    def remove(self,id):
        for x in self.pos:
            if x.emp_id() == id:
                self.pos.remove(x)
    def display(self):
        for x in self.pos:
            print("Name: ",x.name())
            print("ID: ",x.emp_id())
            print("Position: ",x.pos())
            print("Salary : ",x.sal())

d = department()
d.add(Employee(178,"Divya","manager",30000))
d.add(Employee(190,"Vijya",'manager',30000))
d.display()
d.remove(178)
d.remove(190)
print("after")
d.display()

```

```

↩ New Employee is added!!
↩ New Employee is added!!
Name: Divya
ID: 178
Position: manager
Salary : 30000
Name: Vijya
ID: 190
Position: manager
Salary : 30000
after

```

4. Student Record System

- Create a Student class with private attributes student_id, name, age, and grades (a list of integers).
- Implement getter and setter methods for name and age.
- Implement a setter method for grades that ensures all grades are within a valid range (e.g., 0-100).

```

class Student_record:
    def __init__(self,student_id,name,age,grades):
        self.__student_id=student_id
        self.__name=name
        self.__age=age
        self.__grades=grades

    def id(self):
        return self.__student_id
    def name(self):
        return self.__name
    def age(self):
        return self.__age
    def grades(self):
        return self.__grades
    def set_name(self,name):
        self.__name = name
    def set_age(self,age):
        self.__age = age
    def set_grade(self,grade):
        self.__grades.clear()
        for x in grade:
            if x>0 and x<100:
                self.__grades.append(x)
s1 = Student_record(456,"Janvi",17,[90,78,65,87])
s2 = Student_record(561,"Divya",18,[78,89,56,78])
s1.set_grade([23,67,45,89])
s1.grades()

```

 [23, 67, 45, 89]

5. Online Shopping System


Create a Product class with private attributes product_id, name, and price. Implement getter and setter methods for product_id. Implement setter methods for name and price that perform validation (e.g., ensure name is not empty and price is positive)

```

class Shopping_system:
    def __init__(self,product_id,name,price):
        self.__product_id=product_id
        self.__name=name
        self.__price=price
    def set_id(self,id):
        self.__id = id
    def id(self):
        return self.__id
    def set_name(self,name):
        if len(name)>0:
            self.__name = name
        else:
            print("Name is empty!")
    def name(self):
        return self.__name
    def set_price(self,price):
        if price>0:
            self.__price = price
        else:
            print("Price is Negative!")
    def price(self):
        return self.__price

p1 = Shopping_system(645,"Wheat",67)
p1.set_price(-45)
p1.set_name("")
p1.name()

```

 Price is Negative!
Name is empty!
'Wheat'

6.** Transport System**

Question: Develop a transport system. Create a base class Transport with common attributes like id, capacity, and methods like start and stop. Create derived classes Bus, Train, and Airplane with specific attributes and methods. Implement method overriding and polymorphism.

Requirements:

- Transport class with id, capacity, start(), and stop().
- Bus class with route_number attribute and pick_up_passengers() method.

- Train class with number_of_coaches attribute and depart() method.
- Airplane class with flight_code attribute and take_off() m

```
class Transport:
    def __init__(self,id,capacity):
        self.id = id
    def start(self,name):
        print(f"Hello Mr/Ms {name}. Your booking is done on date {self.date}. Your booking id is {self.id}")
    def stop(self,name):
        print(f"Hello {name}. Your booking is cancel.")
class Bus(Transport):
    def __init__(self,id,capacity,route):
        super().__init__(id,capacity)
        self.r = route
    def pickup_passengers(self,name):
        print(f"Welcome to {self.name},")
        super().confirm_booking(name)
        print("enjoy our service!")
class Train(Transport):
    def __init__(self,id,capacity,number_of_coaches):
        super().__init__(id,capacity)
        self.coaches=number_of_coaches
    def depart(self,name):
        print(f"Welcom to {self.no},")
        super().confirm_booking(name)
        print("Thank you for using our service.")
class airplane(Transport):
    def __init__(self,id,capacity,code):
        super().__init__(id,capacity)
        self.code =code
    def take_off(self,name):
        print(f"Welcome in {self.name},")
        super().confirm_booking(name)
b1 = Transport(456,"67")
b3 = Transport(678,"78")
```