Encapsulation :- Encapsulation is the process of hiding any attributes and method(s) from user during the execution of program.

→ We basically uses the concept of 'encapsulation' to prohibit any illegal changes from user.

for this we have to use '__' (double underscore) before the name of 'method' and attributes.

Ex:- __attribute1

__method1():

This is way how python method (_Atm__attributeName) know the method or attribute is encapsulated.

use of getter and set method.

→ __, get() & set() is combindly uses for 'encapsulation'

```
def __init__(self):
    self.__pin = int(input("Enter pin: "))
    self.__balance = 0
```

(attribute to encapsulate)

(attribute to encapsulate but not completely encapsulate without writing get, set method)

```
self.__menu()
```

```
def get_pin(self):
    return self.__pin  (it basically use to see the 'pin' value which is entered)

def set_pin(self, new_pin):
    if type(new_pin) == str:
        self.__pin = new_pin
        print("pin changed")
    else:
        print("not allowed")

def menu(self):
    user_input = input("""Hello, how would u like to proceed?
1. Enter 1 to createpin
2. "  2 " deposit
3. "  3 " withdraw
4. "  4 " Check_bala
5. "  5 " is to exit""")

    if user_input == "1":
        self.create_pin()
    elif user_input == "2":
        self.deposit()
    elif user_input == "3":
        self.withdraw()
    elif user_input == "4":
        self.check_balance()
    else:
        print("bye-bye")

def create_pin(self):
    self.__pin = input("enter ur pin")
    print("Pin set successfully")
```

```
def deposit (self):
    temp = input ("Enter ur pin")
    if temp == self.__pin:
        amount = int (input ("Enter the amount"))
        self.__balance = self.__balance + amount
        print ("Deposit Sucessfull")
    else:
        print ("Invalid Pin")

def withdraw (self):
    temp = input ("Enter ur pin")
    if temp == self.__pin:
        amount = int (input ("Enter the
                 amount"))
        if amount <= self.__balance:
            self.__balance = self.__balance
                             amount
            print ("withdraw Sucessfull")
        else:
            print ("insufficint balance")

    else:
        print ("Invalid pin")

def check_balance (self):
    temp = input ("Enter ur pin")
    if temp == self.__pin:
        print (self.__balance)
    else:
        print ("invalid pin")

sbi = Atm()
```

Abstraction :- Abstraction is the process of hiding the functionality of 'method' ~~data~~.

for full fill the purpose of 'abstraction' we have to add this above the 'Abstracted class'

```
from abc import ABC, abstractmethod
```

- abc → Predefined module
- ABC → class in 'abc' module
- abstractmethod → method

> We can not create 'object' of abstracted class or class that inherit 'Abstract class' of predefined 'Abc' module.

Ex:-

```
from abc import ABC, abstractmethod        (abstract base class)

class BankApp(ABC):                        -> [Inherit abstract class 'ABC' from 'abc' module]
    def database(self):
        print("Connected to database")

    @abstractmethod                        -> (use decorator method '@abstractmethod' to increase its functionality)
    def Security(self):
        pass

    @abstractmethod                        -> (use decorator to increse functify
    def display(self):
        pass

class MobileApp(BankApp):
    def mobile_login(self):
        print("mobile login into mobile")

    def security(self):
        print("mobile Security")

    def display(self):
        print("display")

mob = MobileApp()
mob.security()      [O/P :- Mobile Security]

Obj = BankApp()     -> We can't create object of abstracted
```

Ex:-

```
from abc import ABC, abstractmethod
class LibraryItem(ABC):
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.checked_out = False

    @abstractmethod
    def check_out(self):
        pass

    @abstractmethod
    def check_in(self):
        pass

class Book(LibraryItem):
    def __init__(self, title, author, num_pages):
        super().__init__(title, author)
        self.num_pages = num_pages

    def check_out(self):
        if not self.checked_out:
            self.checked_out = True
            print(f"{self.title} by {self.author} checked out sucessfully.")
        else:
            print("This book is already checkedout")

    def check_in(self):
        if self.checked_out:
            self.checked_out = False
            print(f"{self.title} by {self.author} chicked in sucessfully")
        else:
            print("This book is not checkedout")
```

```
class DVD(LibraryItem):
    def __init__(self, title, director, duration):
        super().__init__(title, director)
        self.duration = duration

    def check_out(self):
        if not self.checked_out:
            self.checked_out = True
            print(f"{self.title} directed by {self.author} checked out successfully")
        else:
            print("This DVD is already checked out")

    def check_in(self):
        if self.checked_out:
            self.checked_out = False
            print(f"{self.title} directed by {self.author} checked in succesfully")
        else:
            print("This DVD is not checked out")

# instance of 'Book' & 'DVD' class
my_book = Book("python", "ipcs it", 404)
my_dvd = DVD("Inception", "pcs embeded", 8

my_book.checkout()
my_dvd.checkout()

my_book.checkin()
my_dvd.check_in()
```

Abstraction

i) Abstraction solve the problem in the design level

ii) Abstraction used to finding unwanted data and give useful data

iii) Abstr" let us focus on what the object does instead of how they does

iv) Outer layout, used in terms of design
For ex:- 'Outer look' of mobile phone, like it has a display screen and keypad to dial a number

Encapsulation

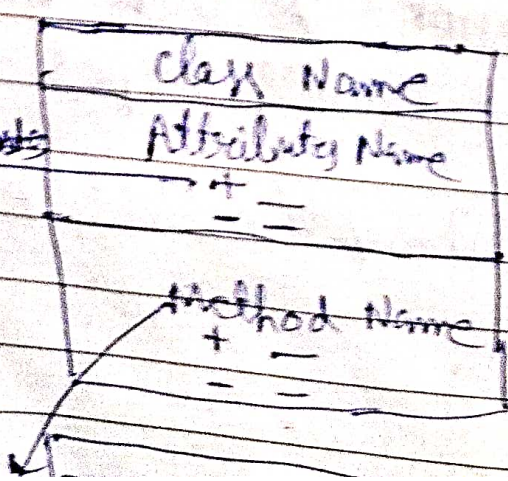
i) Encaps" solve the problem in the implementation level

ii) Encaps" hide data and code from outside world from their unwanted viewing or access.

iii) Encaps" means hiding internal details so that the object or method do something.

iv) Inner layout, used in the terms of implementation
For ex:- 'Inner' implementation detail of a mobile phone like how keypad and display connected

Class Diagram



(-) private attribute

(+) public