In [4]:
```python
# Banking System (Encapsulation and Private Attributes)
class BankAccount:
    def __init__(self,account_number):
        self.__balance=0
        self.__account_number=account_number

    def deposit(self,amount):
        if amount>0:
            self.__balance+=amount
            print(f"deposited ${amount} successfully")
        else:
            print("deposit a positive number")

    def withdraw(self,amount):
        if amount>0 and self.__balance>=amount:
            self.__balance-=amount
            print(f"amount withdrawn successfully ")
        else:
            print("insufficient balance")

    def get_balance(self):
        return self.__balance

    def transfer_money(self,target_account,amount):
        if isinstance(target_account,BankAccount) and amount>0:
            if self.__balance>=amount:
                self.__balance-=amount
                target_account.deposit(amount)
                print(f" amount ${amount} transferred to {target_account.__accou
            else:
                print("insufficient balance")
        else:
            print("invalid amount")

acc1=BankAccount(1002)
acc1.deposit(1000)
acc1.withdraw(500)

acc2=BankAccount(1003)
acc1.transfer_money(acc2,500)
acc1.get_balance()
```

```
deposited $1000 successfully
amount withdrawn successfully
deposited $500 successfully
 amount $500 transferred to 1003
```

Out[4]: 0

In [8]:
```python
#Product Pricing System (Property Decorators)
class Product:
    def __init__(self,price):
        self._price=price
        self.price=price

    @property
    def price(self):
        return self._price
    @price.setter
    def price(self,value):
        if value>0:
            self._price=value
        else:
            raise ValueError ("enter a positive number")

    @price.deleter
    def price(self):
        print("deleting price")
        self._price=None

p=Product(1000)
print("the price of product is :",p.price)

try:
    p.price=-1000
except ValueError as e:
    print("error:",e)

del p.price
print("after deletion the price of the product is ",p.price)
```

```
the price of product is : 1000
error: enter a positive number
deleting price
after deletion the price of the product is  None
```

In [13]:

```
#Employee Salary Management (Abstraction)
from abc import ABC, abstractmethod

class Employee:
    def __init__(self,empid,name):
        self.empid=empid
        self.name=name

    @abstractmethod
    def calculate_salary(self):
        pass

    @abstractmethod
    def raisesalary(self,percentage):
        pass

    @abstractmethod
    def employee_details(self):
        pass

class FullTimeEmployee(Employee):
    def __init__(self,empid,name,monthly_salary):
        super().__init__(empid,name)
        self.monthly_salary=monthly_salary

    def calculate_salary(self):
        return self.monthly_salary

    def employee_details(self):
        print(f"employee name {self.name}, employee id is {self.empid} and emplc

    def raisesalary(self,percentage):
        self.monthly_salary+=self.monthly_salary*percentage/100

class PartTimeEmployee(Employee):
    def __init__(self,empid,name,hourly_rate,hours_worked):
         super().__init__(empid,name)
         self.hourly_rate=hourly_rate
         self.hours_worked=hours_worked
    def calculate_salary(self):
         return self.hourly_rate*self.hours_worked

    def employee_details(self):
        print(f"employee name {self.name}, employee id is {self.empid} and emplo

    def raisesalary(self,percentage):
        self.hourly_rate+=self.hourly_rate*percentage/100

full_time_employee=FullTimeEmployee(101,"uzma",60000)
full_time_employee.employee_details()
full_time_employee.raisesalary(20)

parttimeemployee=PartTimeEmployee(102,"aima",1000,60)
parttimeemployee.employee_details()
parttimeemployee.raisesalary(5)
```

```
employee name uzma, employee id is 101 and employee salary is 60000
employee name aima, employee id is 102 and employee salary is 60000
```

In [ ]: