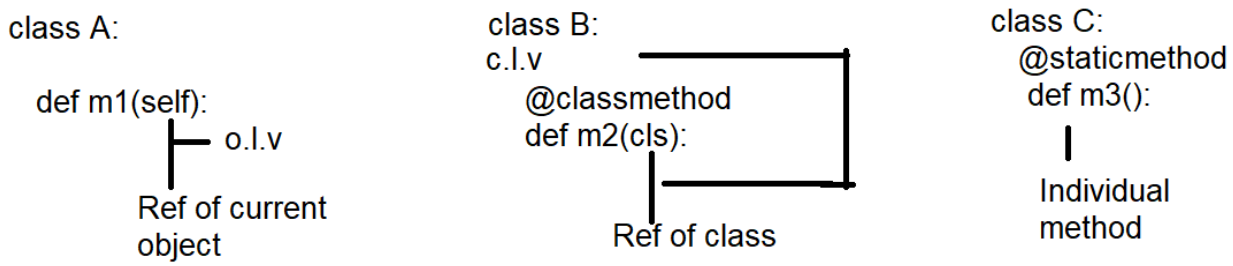
static method

A method defined inside class without implicit first parameter (self, cls) is called static method.
@staticmethod decorator is used to define a method as static.
static methods are global methods, it used to perform global operations.



Syntax:

```
class <class-name>:
        @staticmethod
        def <method-name>():
                statement-1
                statement-2
```

This method is bind with class name and can be invoked without creating object.

Example

```
class A:
    def m1(self):
        print("instance method or object level method")

    @staticmethod
    def m2():
```

```
        print("static method")



obj1=A()
obj1.m1()
A.m2()
```

Output

instance method or object level method
static method

Example:

```
class A:

    def __init__(self):
        self.__x=100
        self.__y=200
    def printXY(self):
        print(f'{self.__x},{self.__y}')

    @staticmethod
    def m1():
        print(self.__x)



A.m1()
```

Output

```
File "C:\Users\nit\PycharmProjects\project1\test25.py", line 11, in m1
    print(self.__x)
          ^^^^
NameError: name 'self' is not defined
```

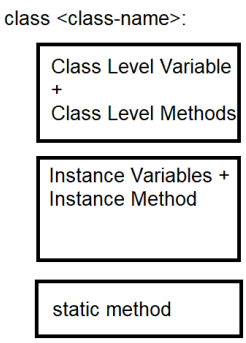
Example

```
class Math:
    @staticmethod
    def power(num,p):
        return num**p
    @staticmethod
    def factorial(num):
        if num==0:
            return 1
        else:
            return num*Math.factorial(num-1)
    @staticmethod
    def isPrime(num):
        c=0
        for i in range(1,num+1):
            if num%i==0:
                c=c+1
        return c==2

res1=Math.power(5,2)
res2=Math.factorial(4)
res3=Math.isPrime(7)
print(res1,res2,res3)
```

Output

```
25 24 True
```



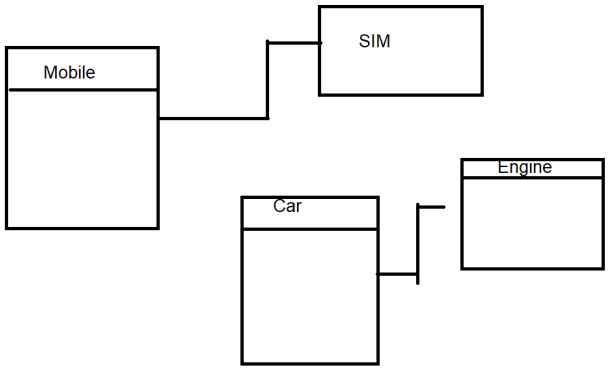
Class Reusability

Object oriented application is collection of classes. The content of one class can be used inside another class in different ways.

1. Composition (Has-A)
2. Aggregation (Special type of composition) (USE-A)
3. Inheritance (IS-A)

Composition

Composition is one of the fundamental concepts in object-oriented programming. It describes a class that references one or more objects of other classes in instance variables.
It is process of creating object of one class inside another class (OR) it is process of a class refers members of another class (OR) a class uses the functionality of another class.



Example:

```
class Engine:
    def start(self):
        print("Engine Start")
    def stop(self):
        print("Engine Stop")

class Car:
    def __init__(self):
        self.__e=Engine()
    def carStart(self):
        self.__e.start()
    def carStop(self):
        self.__e.stop()


car1=Car()
car1.carStart()
car1.carStop()
```

Output

Engine Start

Engine Stop

Example:

```
class Address:
    def __init__(self):
        self.__hno=None
        self.__street=None
        self.__city=None
    def readAddress(self):
        self.__hno=input("HouseNumber :")
        self.__street=input("Street :")
        self.__city=input("City :")
    def printAddress(self):
        print(f'HouseNo {self.__hno}\nStreet
{self.__street}\nCity{self.__city}')


class Person:
    def __init__(self):
        self.__name=None
        self.__add=Address()
    def readPerson(self):
        self.__name=input("Name :")
        self.__add.readAddress()
    def printPerson(self):
        print(f'Name {self.__name}')
        self.__add.printAddress()


p1=Person()
p1.readPerson()
p1.printPerson()
```

Output

Name :naresh

HouseNumber :101

Street :ameerpet

City :hyd

Name naresh

HouseNo 101
Street ameerpet
Cityhyd

Example:

```
class Dept:
    def __init__(self):
        self.__depno=None
        self.__dname=None
    def readDept(self):
        self.__depno=int(input("DeptNo :"))
        self.__dname=input("DeptName :")
    def printDept(self):
        print(f'DeptNo {self.__depno}\nDeptName {self.__dname}')


class Employee:
    def __init__(self):
        self.__empno=None
        self.__ename=None
        self.__dept=[]
    def readEmployee(self):
        self.__empno=input("EmployeeNo :")
        self.__ename=input("EmployeeName :")
        n=int(input("How many dept?"))
        for i in range(n):
            d=Dept()
            d.readDept()
            self.__dept.append(d)

    def printEmployee(self):
        print(f'EmployeeNo {self.__empno}')
        print(f'EmployeeName {self.__ename}')
        for d in self.__dept:
            d.printDept()
```

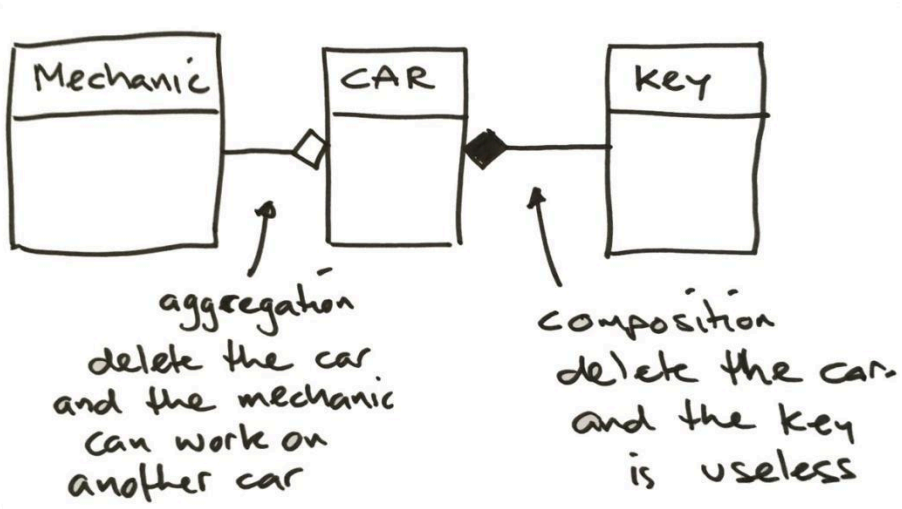
```
emp1=Employee()
emp1.readEmployee()
emp1.printEmployee()
```

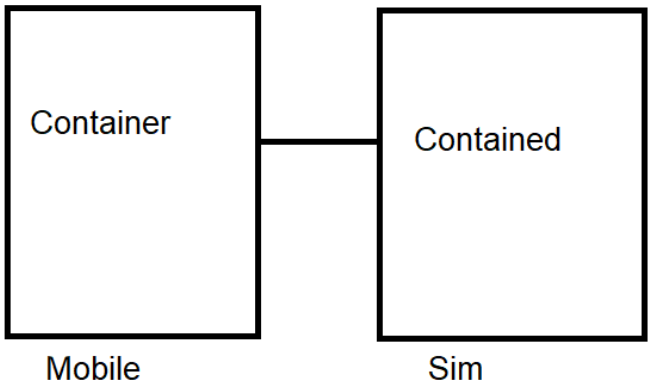
Output

```
EmployeeNo :101
EmployeeName :naresh
How many dept?2
DeptNo :10
DeptName :HR
DeptNo :20
DeptName :Sales
EmployeeNo 101
EmployeeName naresh
DeptNo 10
DeptName HR
DeptNo 20
DeptName Sales
```

Aggregation

Aggregation is a special type of composition.
In aggregation contained object exists independent of container object.


Mechanic
CAR
Key
aggregation
delete the car
and the mechanic
can work on
another car
composition
delete the car.
and the key
is useless


Container
Contained
Mobile
Sim