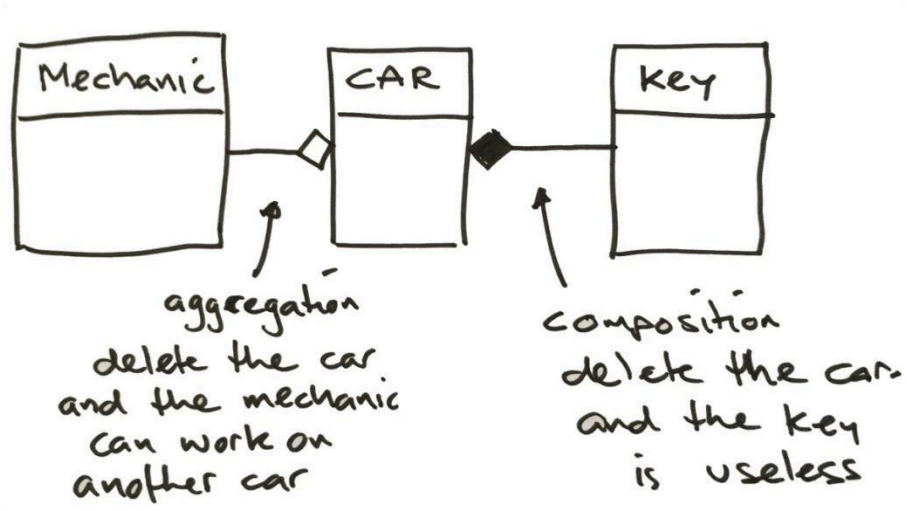
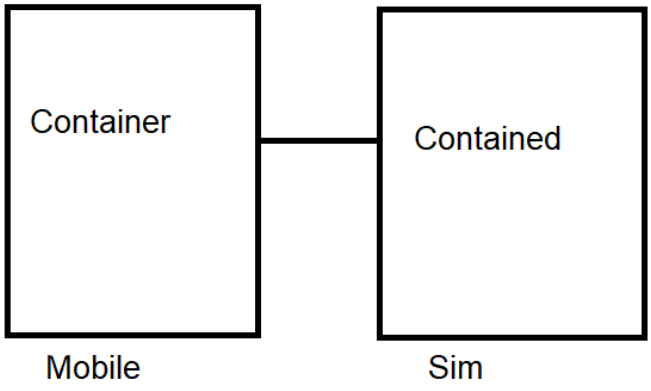
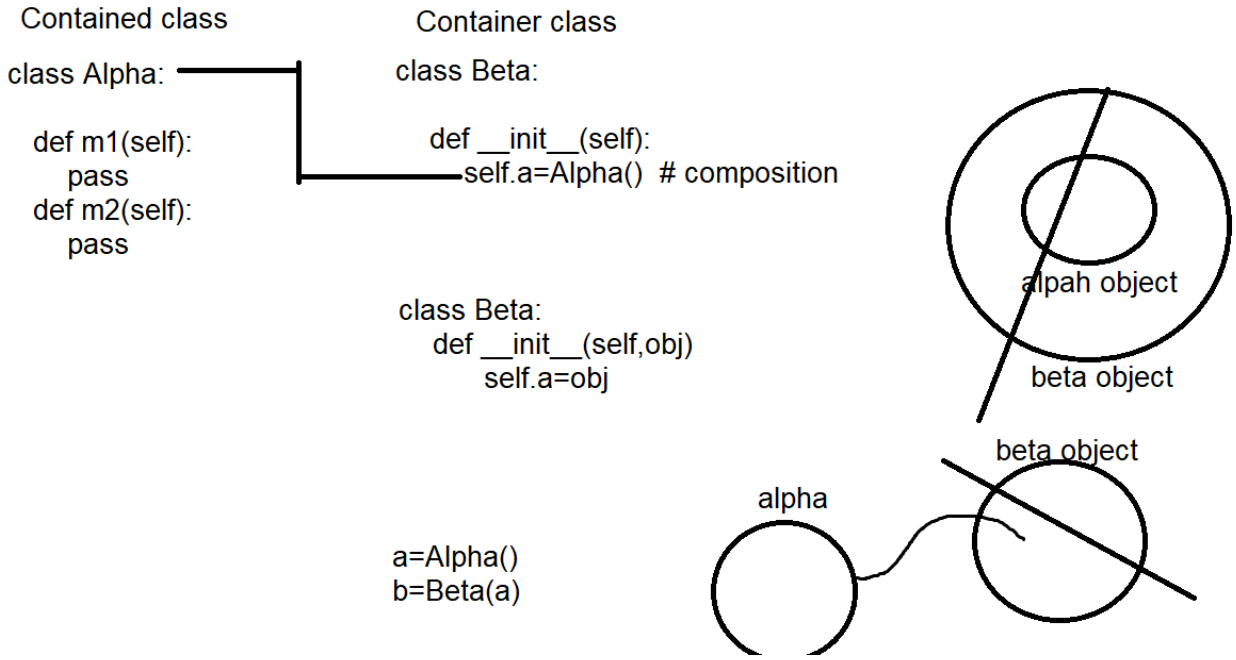
Aggregation

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







In aggregation contained object is send to container object using constructor or method.
In aggregation contained object is not created inside container object but it send to container object using constructors or methods.
In aggregation contained and container objects exists independently.

Example:

```
class Dept:
    def __init__(self,dno,dn):
        self.__deptno=dno
        self.__dname=dn
    def printDept(self):
        print(f'DeptNo {self.__deptno}\nDeptName {self.__dname}')

class Employee:
    def __init__(self,eno,en,d):
        self.__empno=eno
        self.__ename=en
```

```
        self.__dept=d

    def printEmployee(self):
        print(f'EmployeeNo {self.__empno}\nEmployeeName
{self.__ename}')
        self.__dept.printDept()


dept1=Dept(10,"HR")
emp1=Employee(101,"naresh",dept1)
emp2=Employee(102,"suresh",dept1)
emp1.printEmployee()
emp2.printEmployee()
```

Output

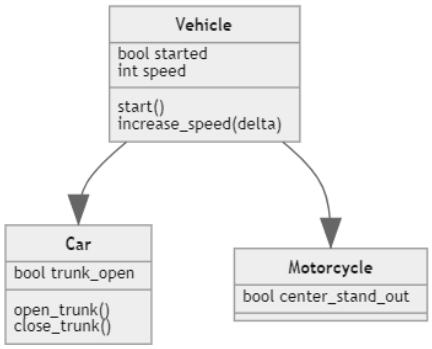
```
EmployeeNo 101
EmployeeName naresh
DeptNo 10
DeptName HR
EmployeeNo 102
EmployeeName suresh
DeptNo 10
DeptName HR
```

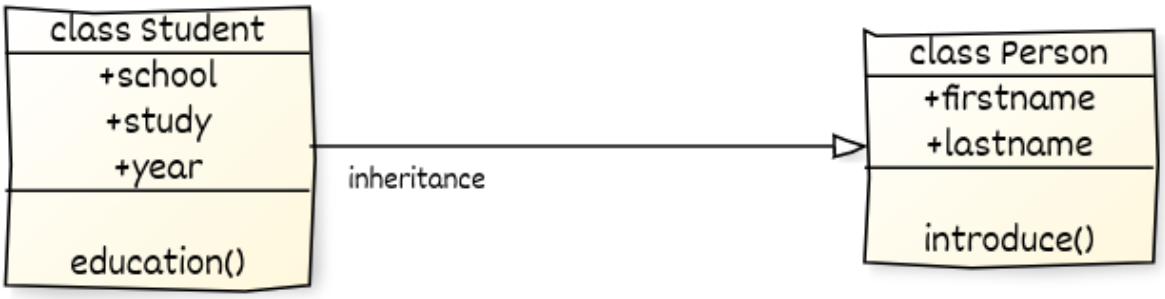
Inheritance

Inheritance allows us to define a class that inherits all the methods and properties from another class. Parent class is the class being inherited from, also called base class. Child class is the class that inherits from another class, also called derived class.

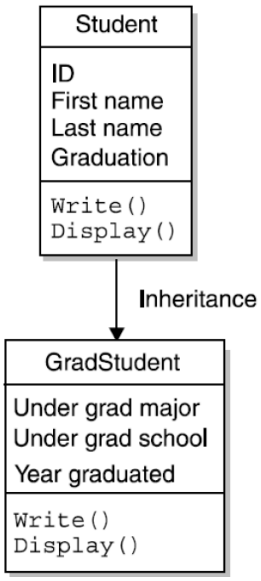
Inheritance is a process of acquiring the properties and behavior of one class inside another class.

Inheritance is process of grouping all the classes which share common properties and behavior.





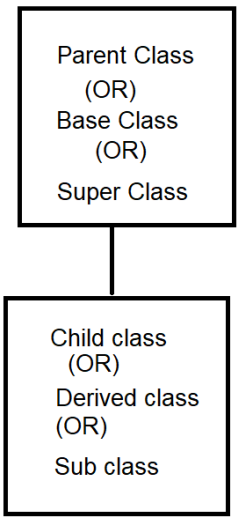
Simple inheritance:



- **Simple inheritance** occurs when there is one parent-child relationship, i.e. one child inherits from one parent.
- Inheritance occurs from the parent to the child. *A parent class cannot access attributes and behaviour of a child class.*
- E.G.: In the situation sketched on the left:

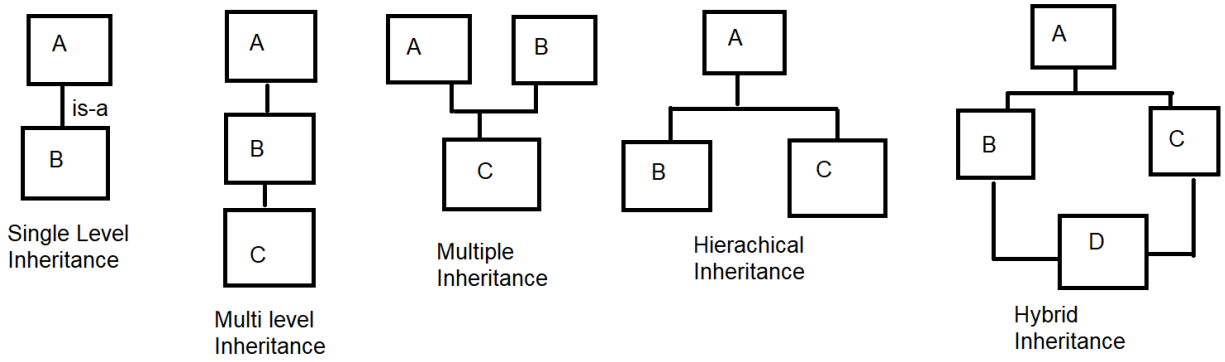
  The 'Student' class cannot call the Write() and Display() methods of the 'GradStudent' class. However, the latter can call the 'Student' class's version of those methods.

Inheritance allows building new data type using existing data type.



Based on the reusability or organization of classes, inheritance are classified into different types.

1. Single level inheritance
2. Multi level inheritance
3. Multiple inheritance
4. Hierarchical inheritance
5. Hybrid inheritance



Syntax of inheritance

class <derived-class-name>(base-class-name,base-class-name,…):
    variables
    methods

Advantage of inheritance is reusability. It allows using the variables and methods of one class inside class.

Derived class is automatically inherits variables and methods of base class.

1. Methods of super class/base class are automatically inherited inside sub class/derived class.

```
class A: # Base class/parent class/super class/generalized class
    def m1(self):
        print("m1 of A")
    def m2(self):
        print("m2 of A")

class B(A): # Derived class/child class/sub class/specialized class
    pass


objb=B()
objb.m1()
objb.m2()
```

Output

m1 of A

m2 of A

Example:

```
class A: # Base class/parent class/super class/generalized class
    def m1(self):
        print("m1 of A")
    def m2(self):
        print("m2 of A")

class B(A): # Derived class/child class/sub class/specialized class
    def m3(self):
        print("m3 of B")
```

```
    def m4(self):
        print("m4 of B")


objb=B()
objb.m1()
objb.m2()
objb.m3()
objb.m4()
```

Output:

m1 of A
m2 of A
m3 of B
m4 of B

2. Properties or variables of base class are not inherited automatically inside derived class.

Example:

```
class A:
    def __init__(self):
        self.x=100
        self.y=200


class B(A):
    def __init__(self):
        super().__init__()
        self.p=300
        self.q=400



objb=B()
```

```
print(objb.x,objb.y)
print(objb.p,objb.q)
```

Output

100 200
300 400

In order to inherit properties of base class within derived class, derived class constructor must call the constructor of base class.

super() function

The super() function is used to give access to methods and properties of a parent or sibling class. The super() function returns an object that represents the parent class.

Syntax:

super()
super(child-class,self)

super() function is used with single level inheritance.