\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Chapter-18\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*18.Inheritance \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

………………………………………………………………………………………………

->Inheritance

->Has-A Relationship

->IS-A Relationship

->IS-A vs HAS-A Relationship

->Composition vs Aggregation

………………………………………………………………………………………………\*\*\*\*\*Using members of one class inside another class:\*\*\*\*:-

………………………………………………………………………………………………

We can use members of one class inside another class by using the following ways

1. By Composition (Has-A Relationship)

2. By Inheritance (IS-A Relationship)

……………………………………………………………………………………

\*\*\*\*1. By Composition (Has-A Relationship):\*\*\*\*\*\*\*\*

……………………………………………………………………………………

By using Class Name or by creating object we can access members of one class inside another class

is nothing but composition (Has-A Relationship).

The main advantage of Has-A Relationship is Code Reusability.

………………………………………………………………………….

Demo Program-1:

1) class Car:

2) def \_\_init\_\_(self,name,model,color):

3) self.name=name

4) self.model=model

5) self.color=color

6) def getinfo(self):

7) print("Car Name:{} , Model:{} and Color:{}".format(self.name,self.model,self.color))

8)

9) class Employee:

10) def \_\_init\_\_(self,ename,eno,car):

11) self.ename=ename

12) self.eno=eno

13) self.car=car

14) def empinfo(self):

15) print("Employee Name:",self.ename)

16) print("Employee Number:",self.eno)

17) print("Employee Car Info:")

18) self.car.getinfo()

19) c=Car("Innova","2.5V","Grey")

20) e=Employee('Prasanna',10000,c)

21) e.empinfo()

……………………………………………………………………………….

Output:

Employee Name: Prasanna

Employee Number: 10000

Employee Car Info:

Car Name: Innova, Model:2.5V and Color:Grey

In the above program Employee class Has-A Car reference and hence Employee class can access all members of Car class.

……………………………………………………………………

Demo Program-2:

1) class X:

2) a=10

3) def \_\_init\_\_(self):

4) self.b=20

5) def m1(self):

6) print("m1 method of X class")

7) class Y:

8) c=30

9) def \_\_init\_\_(self):

10) self.d=40

11) def m2(self):

12) print("m2 method of Y class")

13) def m3(self):

14) x1=X()

15) print(x1.a)

16) print(x1.b)

17) x1.m1()

18) print(Y.c)

19) print(self.d)

20) self.m2()

21) print("m3 method of Y class")

22) y1=Y()

23) y1.m3()

……………………………………………………………..

Output:

10

20

m1 method of X class

30

40

m2 method of Y class

m3 method of Y class

………………………………………………………………………………………..

\*\*\*\*\*\*\*2. By Inheritance(IS-A Relationship):\*\*\*\*\*\*\*\*\*\*\*\*:-

…………………………………………………………………………………………..

What ever variables, methods and constructors available in the parent class by default available to the child classes and we are not required to rewrite. Hence the main advantage of inheritance is Code Reusability and we can extend existing functionality with some more extra functionality.

………………………………………………………………………………………………Syntax :

class childclass(parentclass):

………………………………………………………………………………………………\*\*\*\*Demo Program for inheritance:\*\*\*\*\*\*\*\*\*\*\*\*\*:-

………………………………………………………………………………………….

1) class P:

2) a=10

3) def \_\_init\_\_(self):

4) self.b=10

5) def m1(self):

6) print('Parent instance method')

7) @classmethod

8) def m2(cls):

9) print('Parent class method')

10) @staticmethod

11) def m3():

12) print('Parent static method')

13)

14) class C(P):

15) pass

16)

17) c=C()

18) print(c.a)

19) print(c.b)

20) c.m1()

21) c.m2()

22) c.m3()

…………………………………………………………………………

Output:

10

10

Parent instance method

Parent class method

Parent static method

…………………………………………………………………………

Eg:

1) class P:

2) 10 methods

3) class C(P):

4) 5 methods

In the above example Parent class contains 10 methods and these methods automatically available to the child class and we are not required to rewrite those methods(Code Reusability)

Hence child class contains 15 methods.

…………………………………………………………

Note:

What ever members present in Parent class are by default available to the child class through inheritance.

…………………………………………………..

Demo Program:

1) class P:

2) def m1(self):

3) print("Parent class method")

4) class C(P):

5) def m2(self):

6) print("Child class method")

7)

8) c=C();

9) c.m1()

10) c.m2()

…………………………………………

Output:

Parent class method

Child class method

Whatever methods present in Parent class are automatically available to the child class and hence on the child class reference we can call both parent class methods and child class methods.

Similarly variables also

1) class P:

2) a=10

3) def \_\_init\_\_(self):

4) self.b=20

5) class C(P):

6) c=30

7) def \_\_init\_\_(self):

8) super().\_\_init\_\_()===>Line-1

9) self.d=30

10)

11) c1=C()

12) print(c1.a,c1.b,c1.c,c1.d)

………………………………………………………………………………………………If we comment Line-1 then variable b is not available to the child class.

……………………………………………

\*\*\*\*\*Demo program for inheritance:\*\*\*\*\*\*\*\*\*\*\*\*:-

…………………………………………………………………………………..

1) class Person:

2) def \_\_init\_\_(self,name,age):

3) self.name=name

4) self.age=age

5) def eatndrink(self):

6) print('Eat Biryani and Drink Beer')

7)

8) class Employee(Person):

9) def \_\_init\_\_(self,name,age,eno,esal):

10) super().\_\_init\_\_(name,age)

11) self.eno=eno

12) self.esal=esal

13)

14) def work(self):

15) print("Coding Python is very easy just like drinking Chilled Beer")

16) def empinfo(self):

17) print("Employee Name:",self.name)

18) print("Employee Age:",self.age)

19) print("Employee Number:",self.eno)

20) print("Employee Salary:",self.esal)

21)

22) e=Employee(' Prasanna', 48, 100, 10000)

23) e.eatndrink()

24) e.work()

25) e.empinfo()

……………………………………………………………………

Output:

Eat Biryani and Drink Beer

Coding Python is very easy just like drinking Chilled Beer

Employee Name: Prasanna

Employee Age: 48

Employee Number: 100

Employee Salary: 10000

………………………………………………………………………………………..

\*\*\*\*\*\*\*\*\*\*\*\*IS-A vs HAS-A Relationship:\*\*\*\*\*\*\*\*\*\*\*\*\*:-

…………………………………………………………………………………………….

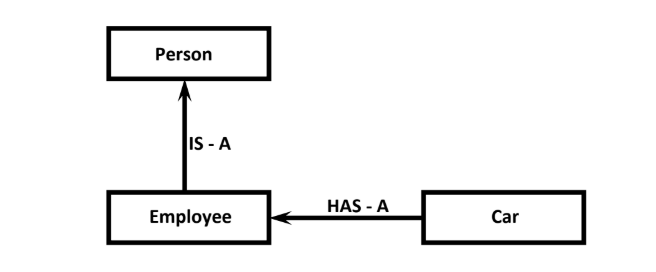
If we want to extend existing functionality with some more extra functionality then we should go for IS-A Relationship

If we dont want to extend and just we have to use existing functionality then we should go for HAS-A Relationship

…………………………………………..

Eg: Employee class extends Person class Functionality

But Employee class just uses Car functionality but not extending



1) class Car:

2) def \_\_init\_\_(self,name,model,color):

3) self.name=name

4) self.model=model

5) self.color=color

6) def getinfo(self):

7) print("\tCar Name:{} \n\t Model:{} \n\t Color:{}".format(self.name,self.model,self.col

or))

8)

9) class Person:

10) def \_\_init\_\_(self,name,age):

11) self.name=name

12) self.age=age

13) def eatndrink(self):

14) print('Eat Biryani and Drink Beer')

15)

16) class Employee(Person):

17) def \_\_init\_\_(self,name,age,eno,esal,car):

18) super().\_\_init\_\_(name,age)

19) self.eno=eno

20) self.esal=esal

21) self.car=car

22) def work(self):

23) print("Coding Python is very easy just like drinking Chilled Beer")

24) def empinfo(self):

25) print("Employee Name:",self.name)

26) print("Employee Age:",self.age)

27) print("Employee Number:",self.eno)

28) print("Employee Salary:",self.esal)

29) print("Employee Car Info:")

30) self.car.getinfo()

31)

32) c=Car("Innova","2.5V","Grey")

33) e=Employee(' Prasanna',48,100,10000,c)

34) e.eatndrink()

35) e.work()

36) e.empinfo()

……………………………………………………………..

Output:

Eat Biryani and Drink Beer

Coding Python is very easy just like drinking Chilled Beer

Employee Name: Prasanna

Employee Age: 48

Employee Number: 100

Employee Salary: 10000

Employee Car Info:

Car Name:Innova

Model:2.5V

Color:Grey

………………………………………………………………………………………………

In the above example Employee class extends Person class functionality but just uses Car class

functionality.

…………………………………………………………………………………………….

\*\*\*\*\*\*\*\*\*\*\*Composition vs Aggregation:\*\*\*\*\*\*\*\*\*\*\*\*:-

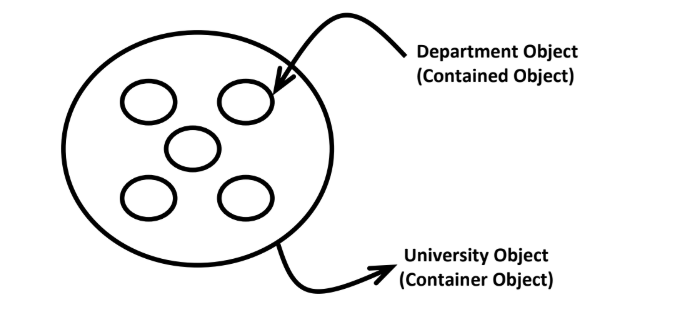
………………………………………………………………………………………

Composition:

Without existing container object if there is no chance of existing contained object then the container and contained objects are strongly associated and that strong association is nothing but Composition.

……………………………………………………………………………………..

Eg: University contains several Departments and without existing university object there is no chance of existing Department object. Hence University and Department objects are strongly associated and this strong association is nothing but Composition.



………………………………………………………………………………………..

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Aggregation:\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

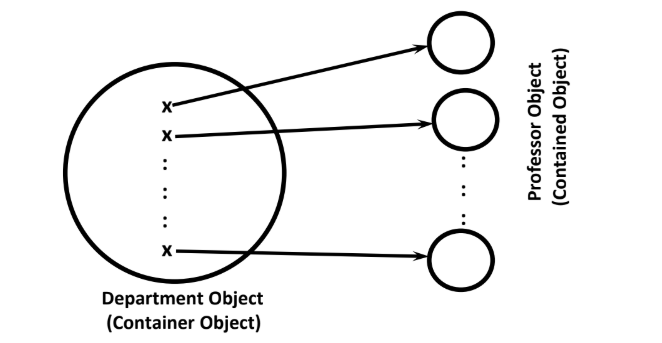
……………………………………………………………………………………….

Without existing container object if there is a chance of existing contained object then the container and contained objects are weakly associated and that weak association is nothing but Aggregation.

……………………………………………………………………

Eg: Department contains several Professors. Without existing Department still there may be a chance of existing Professor. Hence Department and Professor objects are weakly associated,

which is nothing but Aggregation.



………………………………………………………………………………………………

\*\*\*\*\*\*\*\*\*\*\*\*\*Coding Example:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

……………………………………………………………………………………………

1) class Student:

2) collegeName='PRASANNASOFT'

3) def \_\_init\_\_(self,name):

4) self.name=name

5) print(Student.collegeName)

6) s=Student(' Prasanna ')

7) print(s.name)

……………………………………………………..

Output:

PRASANNASOFT

Prasanna

……………………………………………………….

In the above example without existing Student object there is no chance of existing his name. Hence Student Object and his name are strongly associated which is nothing but Composition.

……………………………………………………….

But without existing Student object there may be a chance of existing collegeName. Hence Student object and collegeName are weakly associated which is nothing but Aggregation.

…………………………………………………..

\*\*\*\*\*\*\*\*\*\*Conclusion:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*:-

……………………………………………………………………………………..

The relation between object and its instance variables is always Composition where as the relation between object and static variables is Aggregation.

……………………………………………………

Note: Whenever we are creating child class object then child class constructor will be executed. If the child class does not contain constructor then parent class constructor will be executed, but parent object won't be created.

……………………………………………

Eg:

1) class P:

2) def \_\_init\_\_(self):

3) print(id(self))

4) class C(P):

5) pass

6) c=C()

7) print(id(c))

………………………………….

Output:

6207088

6207088

………………………………………………………………………………………………

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

………………………………………………………………………………………………