Aggregation ¶

- If the link between two objects is weaker, and neither object has exclusive ownership of the other, it can be called aggregation.
- Class B forms an aggregation relationship with Class A, as it references an independent A object when initialized, as one of its attributes. While a B object is dependent on A, in the event of B's destruction, a will continue to exist as it is independent of B.

In [1]:

```
class A(object):
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def addNums():
        self.b + self.c

class B(object):
    def __init__(self, d, e, A):
        self.d = d
        self.e = e
        self.A = A

    def addAllNums(self):
        x = self.d + self.e + self.A.b + self.A.c
        return x

Obj_A = A("Class A", 2, 6)
Obj_B = B(5, 9, Obj_A)

print (Obj_B.addAllNums())
```

```
22
```

Composition

- Composition is a way of aggregating objects together by making some objects attributes of other objects.
- According to some formal definitions the term composition implies that the two objects are quite **strongly linked** – one object can be thought of as belonging **exclusively** to the other object. If the owner object ceases to exist, the owned object will probably cease to exist as well.

In [2]:

```
class A(object):
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    def addNums():
        self.b + self.c

class B(object):
    def __init__(self, d, e):
        self.d = d
        self.e = e
        self.A = A("Class A", 2, 6)

    def addAllNums(self):
        x = self.d + self.e + self.A.b + self.A.c
        return x

Obj_B = B(5, 9)

print (Obj_B.addAllNums())
```

```
22
```

Much like aggregation, however rather than referencing an independent object, B actually initializes an instance of A in it's own constructor as an attribute. If the B object is destroyed then so too is the A object. This is why composition is such a strong relationship.

In [ ]:

```

```