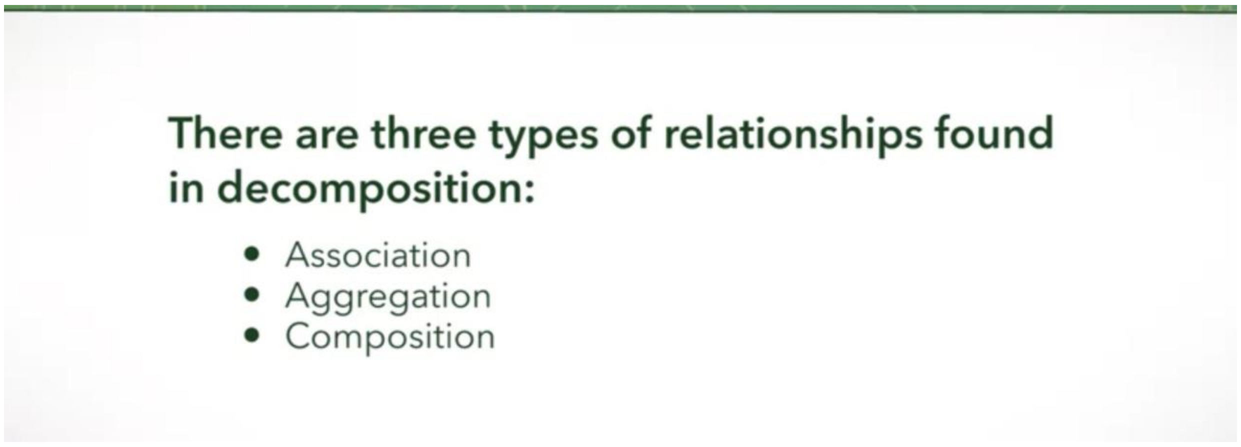
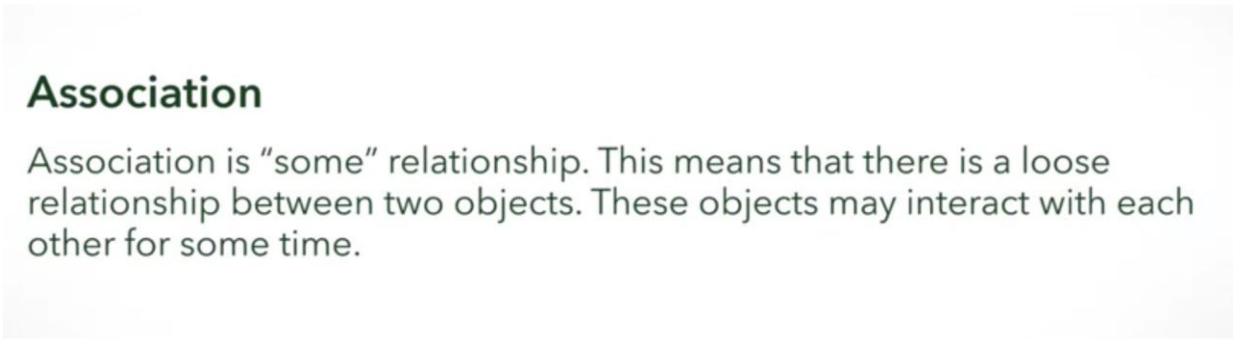


There are three types of relationships found in decomposition:

- Association
- Aggregation
- Composition

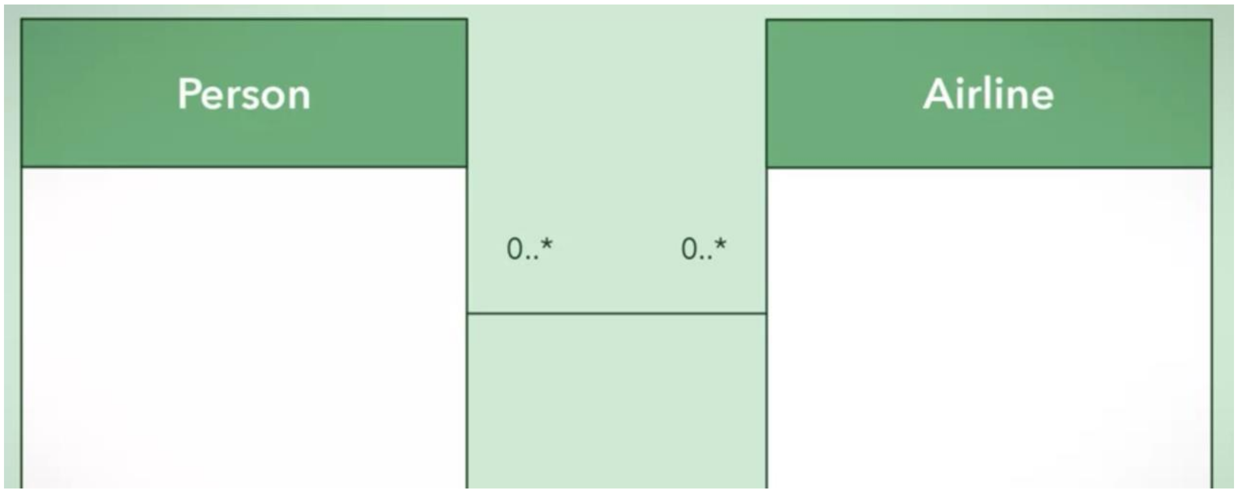


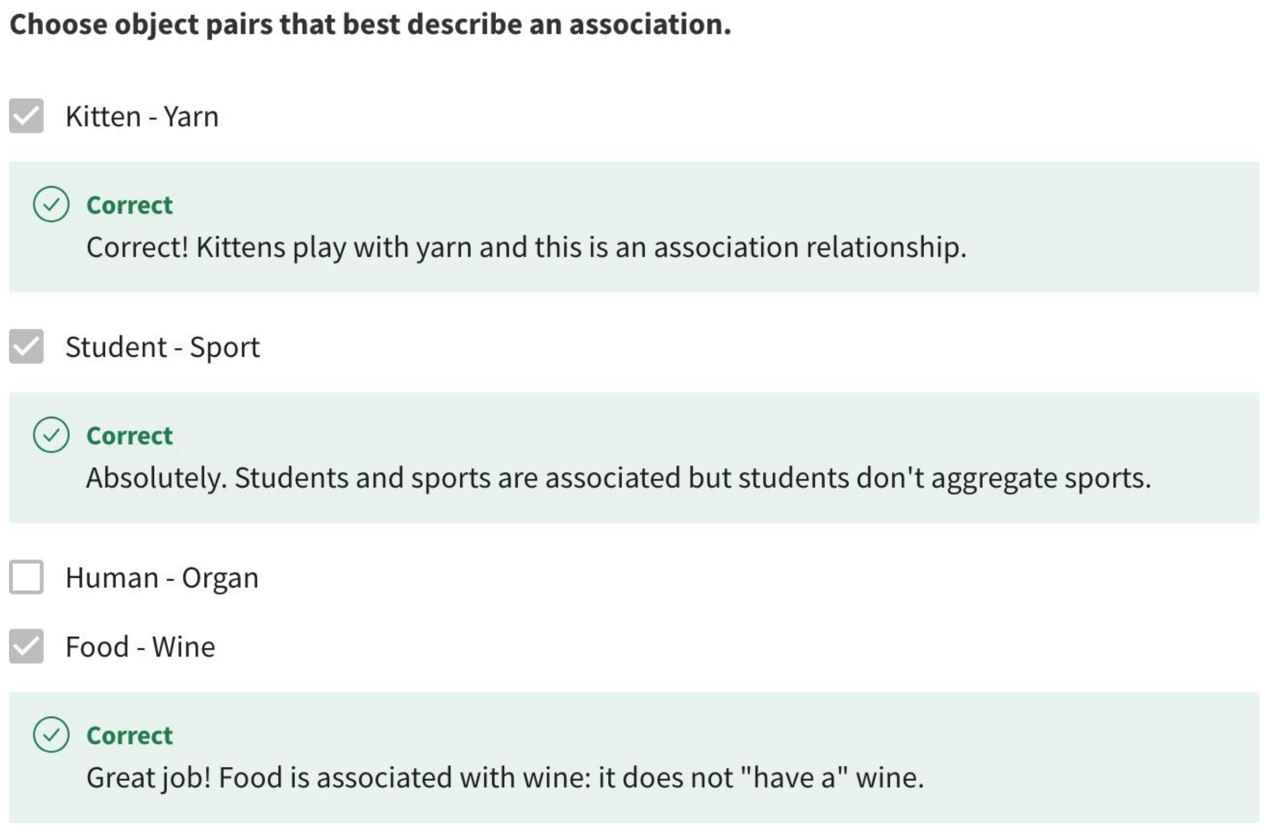
Association

Association is “some” relationship. This means that there is a loose relationship between two objects. These objects may interact with each other for some time.

For example, an object of a class may use services/methods provided by object of another class. This is like the relationship between person and airline. A person does not generally own an airline, but can interact with one. An airline can also interact with many person objects. There are some persons and some airlines, neither is dependent on the other.

This means a person object is associated with zero or more airline objects. And an airline object is associated with zero or more person objects.





Choose object pairs that best describe an association.

[✓] Kitten - Yarn

Correct
Correct! Kittens play with yarn and this is an association relationship.

[✓] Student - Sport

Correct
Absolutely. Students and sports are associated but students don't aggregate sports.

☐ Human - Organ

[✓] Food - Wine

Correct
Great job! Food is associated with wine: it does not "have a" wine.

. If one object is destroyed, the other can continue to exist, unlike human and organ. One object does not belong to another.

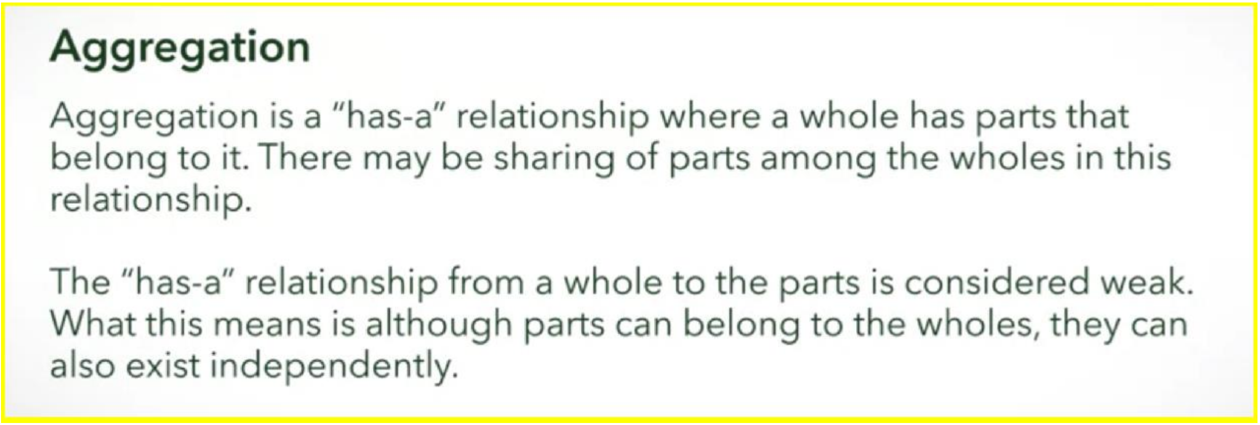
Here No object embedded with other class , it is loosely , separate entity, Kindly below code

```
Class Wine {
            public :
             Void makepair(Food fd ) {
        }
}
```

The wine can exist independent of the food. It does not need food to exist, nor does it always have food.

Overall, association is a loose partnership between two objects that exist completely independently.

==

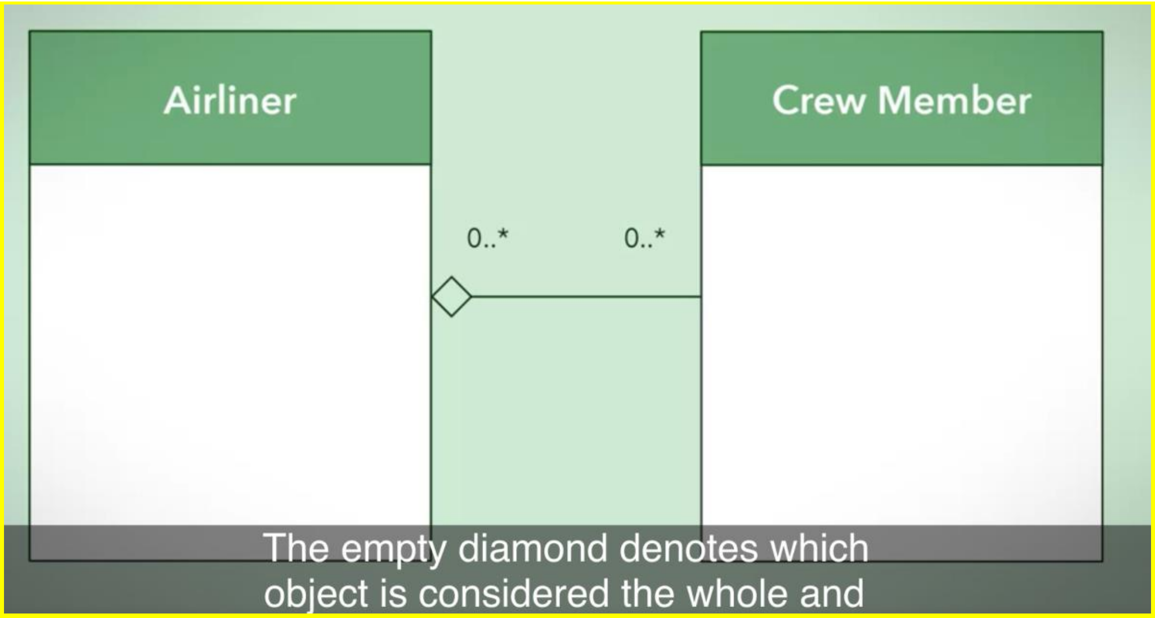


Aggregation

Aggregation is a "has-a" relationship where a whole has parts that belong to it. There may be sharing of parts among the wholes in this relationship.

The "has-a" relationship from a whole to the parts is considered weak. What this means is although parts can belong to the wholes, they can also exist independently.

Aggregation is a weak has-a relationship between classes. One object has the other, but the objects are not heavily linked.


Airliner
Crew Member
0..*
0..*
The empty diamond denotes which
object is considered the whole and

. If one of the objects in the relationship is destroyed, it still makes sense that the other can continue to exist.

Aggregation (Weak Ownership)

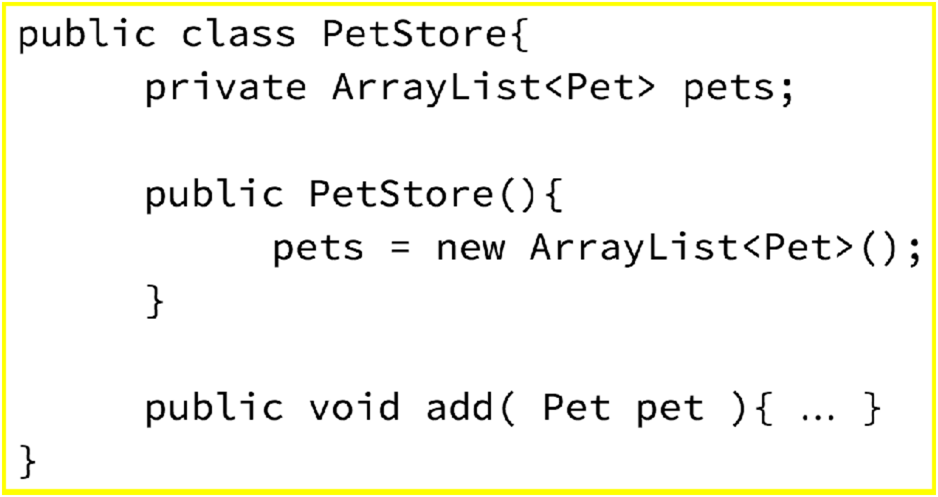
Aggregation is a "has-a" relationship where one object contains another, but the contained object can exist **outside** the container.

Only used pointer, two class are independent

- **Ownership:** Weak. The container doesn't "own" the life of the child.
- **Lifetime:** If the container is destroyed, the child objects continue to exist.
- **C++ Implementation:** Pointers or `std::weak_ptr`.

```
class Teacher {
public:
    std::string name;
    Teacher(std::string n) : name(n) {}
};

class Department {
    // Aggregation: The Department has Teachers,
    // but if the Department closes, Teachers still exist.
    Teacher* HODTeacher;
public:
    Department(Teacher* t) : HODTeacher(t) {}
};
```



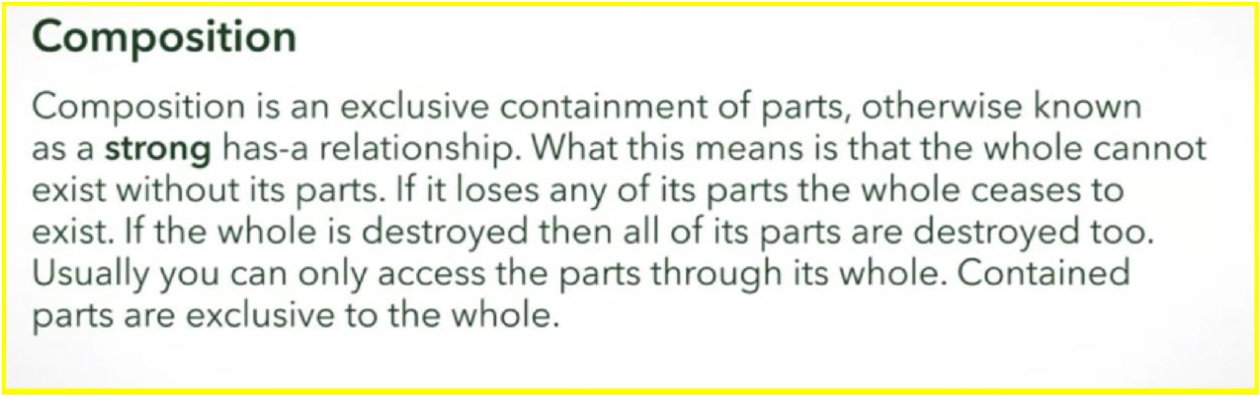
```
public class PetStore{
	private ArrayList<Pet> pets;

	public PetStore(){
		pets = new ArrayList<Pet>();
	}

	public void add( Pet pet ){ … }
}
```

These are two objects that have a weak has-a relationship. The pet store has a list of pets that can contain zero or more pets. It has the ability to add pets at any time. Both pets stores and pets can exist without each other.

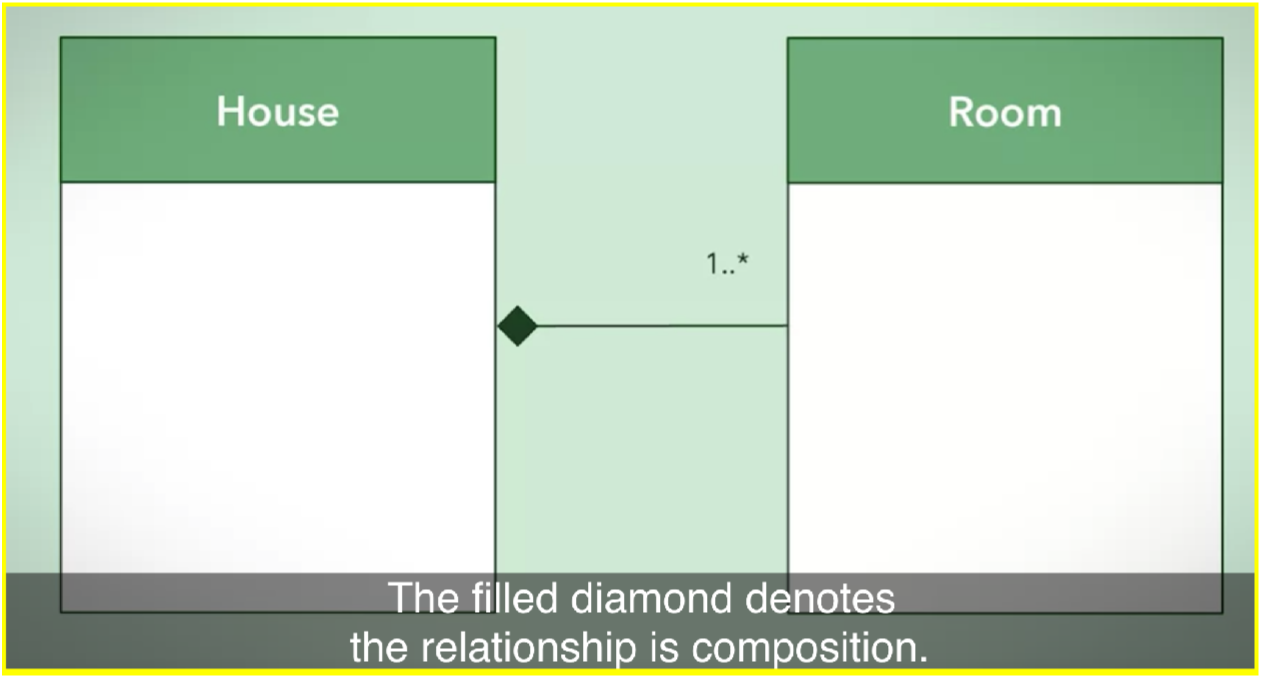
==



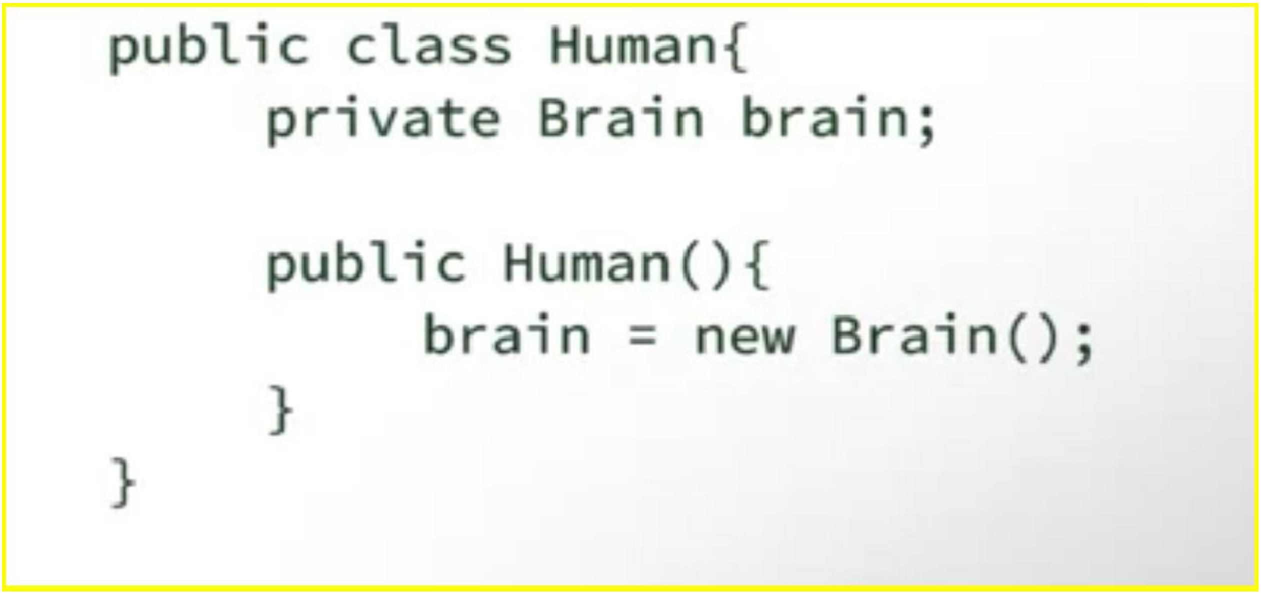
Composition

Composition is an exclusive containment of parts, otherwise known as a **strong** has-a relationship. What this means is that the whole cannot exist without its parts. If it loses any of its parts the whole ceases to exist. If the whole is destroyed then all of its parts are destroyed too. Usually you can only access the parts through its whole. Contained parts are exclusive to the whole.

Composition known as a strong has-a relationship.



The composition is between the human and brain pair. This relationship is between completely dependent classes. **If one object is destroyed, then the other is too.**



```
public class Human{
        private Brain brain;

        public Human(){
                brain = new Brain();
        }
}
```

```
public class Employee{
 private Salary salary;
  public Employee(Salary employeeSalary){
   this.salary = employeeSalary;
 }
}
```

Summary :-

You can use association, a very loose interaction between two completely independent objects.
An aggregation, one whole has a part, but both can live independently.
And finally, in composition, the whole cannot exist without its parts and vice versa. All three relationships are useful and versatile for your software designs.