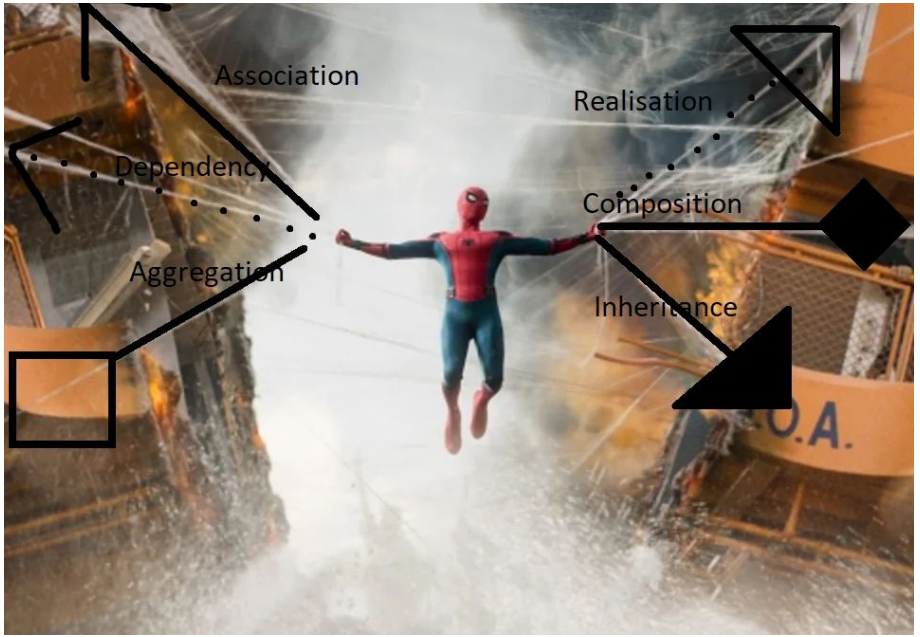
Association



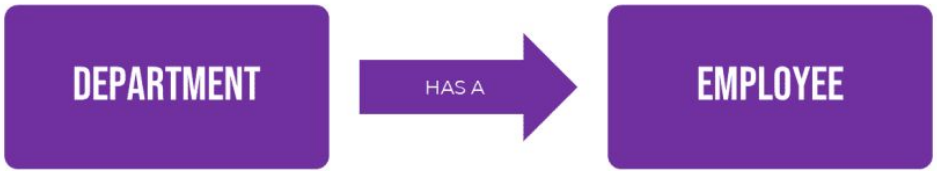
Has A Relationship

Sometimes, one class has a data member which is a object of another class.

A car has a engine

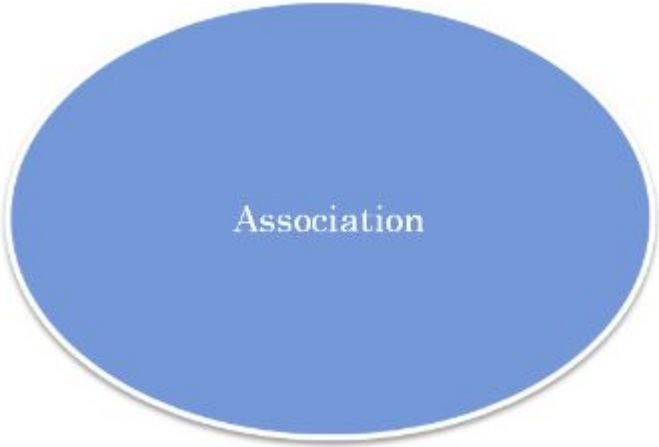
Child have a pet

Customer has a orders



Association

Association is the most general type of relationship and it includes other types as well. If the relationship doesn't fall into a more specific type, like Aggregation or Composition, it can simply be referred to as an Association.



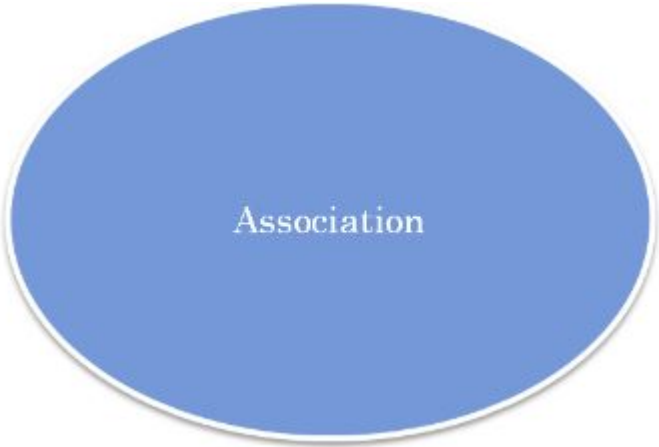
Association

It can be one to one.

It can be one to many.

It can be many to many.

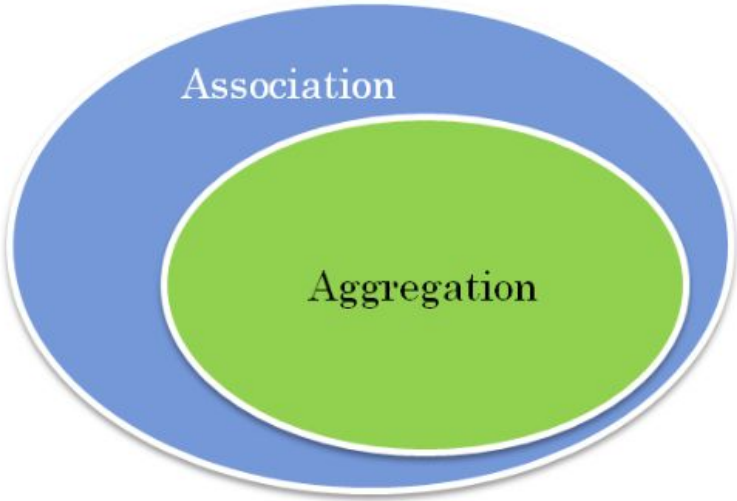
It can be many to one.



Aggregation

Aggregation is a more specific type of association.

In an aggregation the object can also be shared with another owner at the same time, and even if the owner no longer exists, the object can still continue their lifetime.



Aggregation

Example: The relationship between a UserGroup and the Users, is an Aggregation. A User can still have meaning in the system even if it doesn't belong to a UserGroup, so if you delete a UserGroup you won't delete its Users. On the other hand, the Users can belong to several UserGroups at the same time.

Aggregation

```
class Address {
    public:
   string addressLine, city, state;
     Address(string addressLine, string city, string state)
    {
        this->addressLine = addressLine;
        this->city = city;
        this->state = state;
    }
};
```

Aggregation

```
class Employee
    {
        private:
        Address * address;  //Employee HAS-A Address
        public:
        int id;  string name;
        Employee(int id, string name, Address* address)
       {
           this->id = id;
           this->name = name;
           this->address = address;
       }
     void display()
       {    cout<<id <<" "<<name<< " "<<
             address->addressLine<< " "<< address->city<< " "<<address->state<<endl;
       } };
```

Aggregation

```
int main(void) {
    Address a1= Address("C-146, Sec-15","Gulshan","KHI");
    Employee e1 = Employee(101,"Ali",&a1);
            e1.display();
   return 0;
}
```

Aggregation

```
class Address {
    public:
   string addressLine, city, state;
     Address(string addressLine, string city, string state)
    {
        this->addressLine = addressLine;
        this->city = city;
        this->state = state;
    }
};
```

Aggregation

```
class Employee
    {
        private:
        Address * address;  //Employee HAS-A Address
        public:
        int id;  string name;
        Employee(int id, string name, Address* address)
       {
           this->id = id;
           this->name = name;
           this->address = address;
       }
       void setAddress(Address * address)
       {
           this->address=address;
       }
     void display()
       {    cout<<id <<" "<<name<< " "<<
             address->addressLine<< " "<< address->city<< " "<<address->state<<endl;
       } };
```

Aggregation

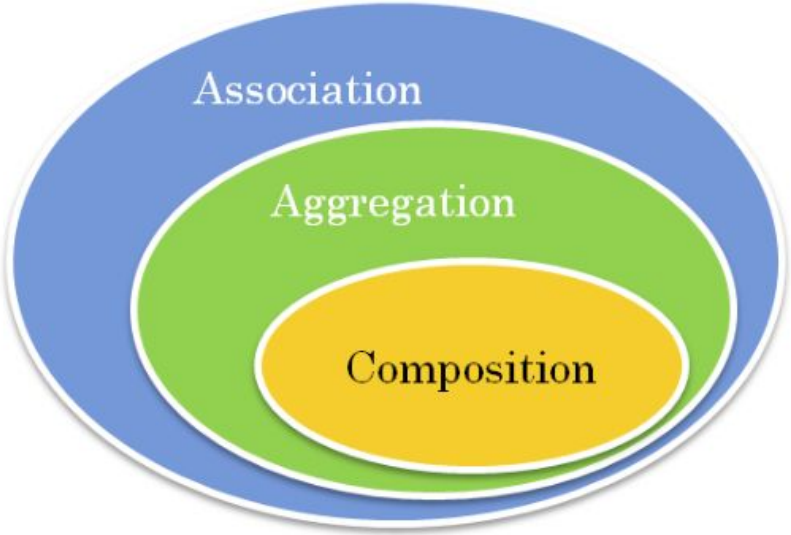
```
int main(void) {
    Address a1= Address("C-146, Sec-15","Karachi","Sindh");
    Address a2= Address("A-146, Sec-10","Lahore","Punjab");
    Employee e1 = Employee(101,"Ali",&a1);
    e1.display();
    e1.setAddress(&a2);
    e1.display();
   return 0;
}
```

Composition

Composition is a more strict type of Aggregation.

In an composition, the object cannot be shared with a different owner and it doesn't make sense for the object to exist without their owner.

So, you usually want to delete the object if you delete their owner.



Composition

The relationship between an Order and the OrderDetails is a composition relationship.

The OrderDetail items are valid only as long as there is an Order related to them. When you delete an Order you will delete it's OrderDetails as well. An OrderDetail associated to a particular Order cannot belong to a different Order.

Composition - Example 1

```
class Employee
    {
        private:
        int id;  string name;
        public:
        Employee(int id, string name)
       {
           this->id = id;
           this->name = name;
       }
       Employee() {}
     void display()
       {    cout<<" "<<name<< " "<<  endl;
       }
};
```

Composition

```
class Department {
public:
Department(string name, string managerName, int managerID,  int maxEmployees)
    : name(name), HoD( managerID,managerName), numEmployees(0),
maxEmployees(maxEmployees) {}

void Display()
{
    cout<<"Department Name "<<name<<endl;
    cout<<"Hod Name";
    HoD.display();
}
private:
    string name; Employee HoD; int numEmployees; int maxEmployees;
};
```

Composition

```
int main() {

    const int maxEmployeesPerDepartment = 15;
    Department CS("Computer Science", "Dr. Ghufran", 1234,
maxEmployeesPerDepartment);
    CS.Display();

   return 0;
}
```

Composition - Example 2

```
class Employee
    {
        private:
        int id;  string name;
        public:
        Employee(int id, string name)
       {
           this->id = id;
           this->name = name;
       }
       Employee() {}
     void display()
       {    cout<<" "<<name<< " "<<  endl;}
     void set (int id, string name)
       {
           this->id = id;
           this->name = name;
       }
    };
```

Composition - Example 2

```
class Department {
    private:
    string name; Employee HoD; int numEmployees; int maxEmployees; Employee
employees[15];
public:

    Department(string name, string managerName, int managerID,  int
maxEmployees)
    : name(name), HoD( managerID,managerName), numEmployees(0),
maxEmployees(maxEmployees) {}
void Display()
{
    cout<<"Department Name "<<name<<endl;
    cout<<"Hod Name";
    HoD.display();
}
```

Composition - Example 2

```
 void addEmployee(string name, int id) {
    if (numEmployees < maxEmployees) {
        employees[numEmployees++] = Employee(id, name);
        // employees[numEmployees++].set(id,name);
    } else {
        cout << "Error: Department is at maximum capacity."
<< endl;
    }
}
```

Composition - Example 2

```
void displayEmployees() {
    cout << "Employees in the " << name << " department:" <<
endl;
    cout << "HoD: ";
    HoD.display();
    for (int i = 0; i < numEmployees; ++i) {
        cout << i<<"-" ;
        employees[i].display();
    }
}};
```

Composition - Example 2

```
int main() {

    const int maxEmployeesPerDepartment = 15;
    Department CS("Computer Science", "Dr. Zulfiqar", 1234,
maxEmployeesPerDepartment);
    CS.Display();
    CS.addEmployee("Sumaiyah",1543);
    CS.displayEmployees();
   return 0;
}
```

Composition - Example 3

```
class Department {
    private:
    string name; Employee HoD; int numEmployees; int
maxEmployees; Employee *employees;
public:

Department(string name, string managerName, int managerID,
int maxEmployees)
    : name(name), HoD( managerID,managerName),
numEmployees(0), maxEmployees(maxEmployees) {
        employees= new Employee[maxEmployees];
    }
```

Class Activity

Make a copy constructor for the above class.

Class Activity

Consider a scenario in which you are designing a social media application. The main entities in this system are User, Post, and Comment. Users can create posts, and other users can leave comments on these posts. How association relationships can be established between these entities, and provide examples of how users, posts, and comments might be interconnected in the system?

Class Activity

A user can have so many posts.

Implement this.

Class Activity

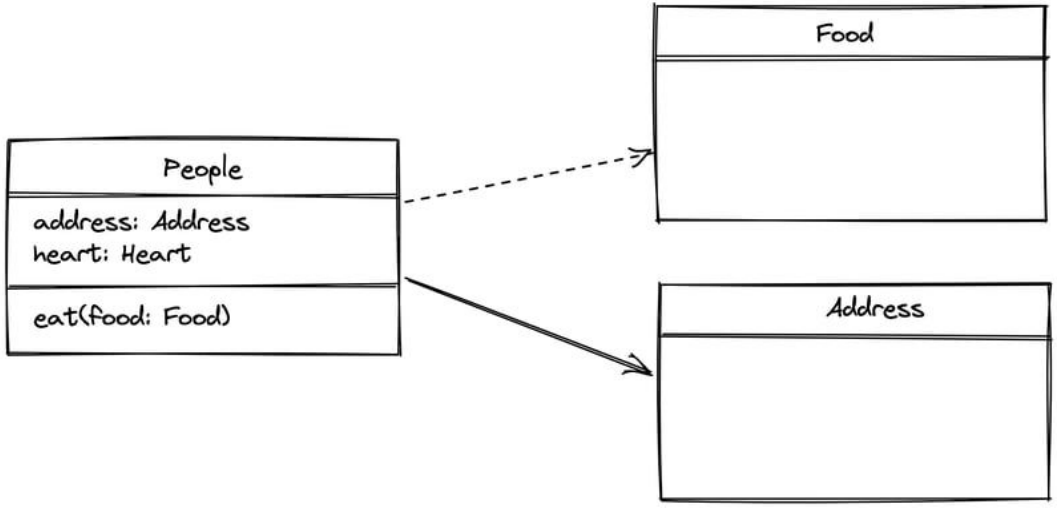
A post can have so many comments.

Implement this.

Dependency

Association --> A has-a C object (as a member variable)

Dependency --> A references B (as a method parameter or return type)



```
class Account {

public:

    Account(int id, double balance) : id(id),
balance(balance) {}

    int getId() { return id; }

    double getBalance() { return balance; }

private:

    int id;

    double balance;

};
```

```
class Bank {

public:

    void transferMoney(Account fromAccount, Account
toAccount, double amount) {

        // Code to transfer money from one account to
another

    } };
```

```
int main() {
    Account account1(123, 1000.00);
    Account account2(456, 500.00);
    Bank bank;
    bank.transferMoney(account1, account2, 250.00);
    return 0;
}
```

Pass By Reference

```
class Account {
public:
    Account(int id, double balance) : id(id), balance(balance) {}
    void subtractBalance(int amount) {balance =balance-amount;}
    void addBalance(int amount) {balance =amount+balance;}
private:
    int id;
    double balance;
};
```

Pass By Reference

```
class Bank {

public:

void transferMoney(Account& from, Account&  to, int amount)
{

      to.addBalance(amount);

      from.subtractBalance(amount);

    } };
```

```
int main() {
    Account account1(123, 1000.00);
    Account account2(456, 500.00);
    Bank bank;
    bank.transferMoney(account1, account2, 250.00);
    return 0;
}
```

Pass By Pointer

```
class Account {
public:
    Account(int id, double balance) : id(id), balance(balance) {}
    void subtractBalance(int amount) {balance =balance-amount;}
    void addBalance(int amount) {balance =amount+balance;}
private:
    int id;
    double balance;
};
```

Pass By Pointer

```
class Bank {

public:

void transferMoney(Account * from, Account *  to, int
amount) {

      to->addBalance(amount);

      from->subtractBalance(amount);

    } };
```

Pass By Pointer

```
int main() {
    Account account1(123, 1000.00);
    Account account2(456, 500.00);
    Bank bank;
    bank.transferMoney(&account1, &account2, 250.00);
    return 0;
}
```

References

https://baharestani.com/2014/08/14/association-vs-aggregation-vs-composition/

https://blog.devgenius.io/association-composition-and-aggregation-in-c-925465987061