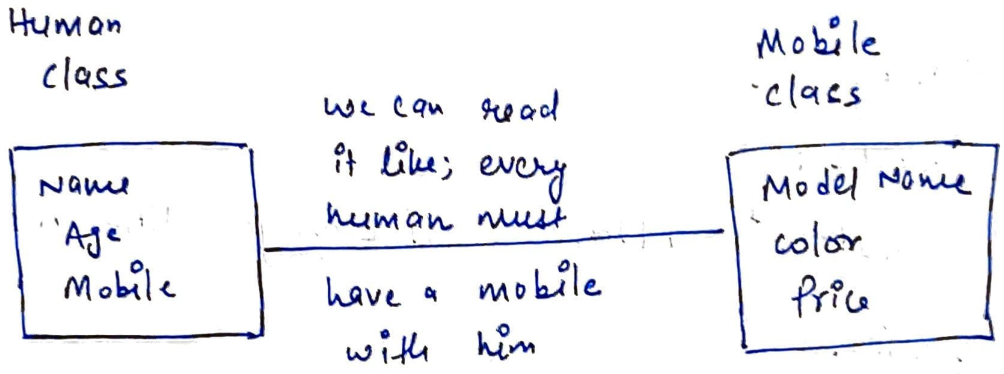
C++ Aggregation (HAS - A Relationship)

In C++, aggregation is a process in which one class defines another class as an entity reference. It is another way to reuse the class. It is a form of association that represents HAS-A relationship.

In simple terms aggregation is when a class is having an object of another class.

example


Human class
Name
Age
Mobile
we can read it like; every human must have a mobile with him
Mobile class
Model Name
color
Price

so whenever we create a human object, it will definitely have a mobile with her. So it is considered as strong relation and technically, this relation is called aggregation.

eg.

Let's see an example of aggregation where Employee class has the reference of Address class as data member. In such way it can reuse the members of

Address class.

```
class Address {
    public:
        string addressLine, city, state;

        Address (adressLine, city, state) {
            this -> addressLine = addressline;
            this -> city = city;
            this -> state = state;
        }
    };

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;
    }
};

int main() {
    Address a1 = Address ("C-146", "Noida", "UP");
    Employee e1 = Employee (101, "Nakul", &a1);
     e1.display();
}
```

Output

```
101  Nakul    C-146  Noida  UP .
```

C++ Polymorphism

The term "polymorphism" is the combination of "poly" + "morphs" which means many forms. It is a greek word. In object oriented programming, we use 3 main concepts : inheritance, encapsulation and polymorphism.

→ Example of Polymorphism

Let's consider a real life example of polymorphism. A lady behaves like a teacher in a classroom, mother or daughter in home and customer in market. Here, a single person is behaving differently according to the situation. This is essentially polymorphism i.e taking many forms.

→ There are 2 types of Polymorphism in C++

$$\begin{tikzcd}
 & \boxed{\text{Compile time polymorphism}} \arrow[r] \arrow[dr] & \boxed{\text{Function overloading}} \\
\boxed{\text{Polymorphism}} \arrow[ur] \arrow[dr] & & \boxed{\text{operator overloading}} \\
 & \boxed{\text{Run time polymorphism}} \arrow[r] & \boxed{\text{Virtual function}}
\end{tikzcd}$$

(1) compile time polymorphism → The overloaded functions are invoked by matching the type and number of arguments. This information is available at compile time and therefore, compiler selects the appropriate function at compile time.

It is achieved by function overloading and operator overloading which is also known as static binding or early binding.

Now, let's consider the case where function name and prototype is same.

eg.

```
class A {
    int a;
    public:
        void display(){
                cout << "Class A" << endl;
        }
};
class B: class A {              // derived class declaration
    int b;
    public:
    void display(){
        cout << "Class B";
    }
};
```

In the above case, the prototype of display function is same in both the base and derived class. Therefore, the static binding cannot be applied. It would be great if the appropriate function is selected at the run time. This is known as run-time polymorphism.

→ Run time polymorphism :- Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time. It is achieved by method overriding which is also known as dynamic binding or late binding.

Compile time polymorphism	Run time polymorphism.
→ The function to be invoked is known at the compile time	The function to be invoked is known at run time.
→ It is also known as overloading, early binding and static binding	It is also known as overriding, dynamic binding and late binding.
→ Overloading is a compile time polymorphism where	Overriding is a run time polymorphism where more

more than one method is having the same name but with the different number of parameters or the type of the parameters	than one method is having the some name, number of parameters and the type of parameters.
→ It is achieved by function overloading and operator overloading.	It is achieved by virtual functions and pointers.
→ It provides fast execution as it is known at the compile time	It provides slow execution as it is known at run time.
→ It is less flexible as mainly all the things execute at the compile time	It is more flexible as all the things execute at the run time.

⇒ C++ Runtime polymorphism example

Let's see an example without the virtual keyword.

```
class Animal {
    public:
        void eat() {
            cout << " Eating...";
        }
};


class Dog : public Animal {
    public:
        void eat() {
            cout << " Eating bread...";
        }
};


int main() {
    Dog d = Dog();
    d.eat();
}
```

<u>output</u>

```
Eating bread...
```

Then, instead of getting Eating... as we would have gotten if we didn't explicitly do Dog d = Dog() at run time.