Encapsulation: Wrapping up data members(properties/states) and functions(methods/behaviors). We wrapped all the data members and functions in single capsule/entity. That capsule is a class.

Fully encapsulated class: All data members are private. Only accessible in same class and not in other class.

Why Encapsulation?

- Encapsulation means data hiding/ information hiding. (by private access modifier)
- If we want then we can make class read only. (by getters)
- Code reusability
- Unit testing

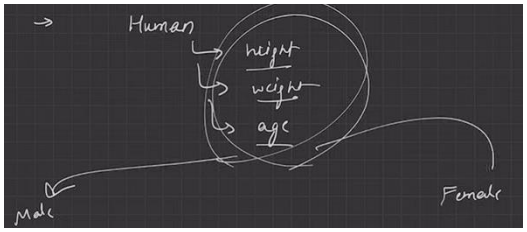
```
class Student {

    // By default private, so don't need to
    private:
        string name;
        int age;
        int height;

    public:
        Codiumate: Options | Test this method
        int getAge() {
            return this -> age;
        }

};
```

It's called encapsulation. Class contain some functions and data members.

Inheritance:

It inherits some properties of one class to another. It's like if your father's height is big then you will also get it.

Human class contain properties like height, weight and age. Male and Female class will also contain properties of human. So, instead of writing all those properties in Male and Female class we can inherit those from Human class.



Super class and sub class

Human class is a parent class/super class as it's properties are inheriting by other classes.

Male and Female classes are sub class or child class as they're inheriting from Human class.

By this, Male class can now able to access all the properties of Human with Male' properties.

Mode of Inheritance:

If you've inherited class in public mode then your answer also be in public mode. When you bring some property of other class by inheriting it, then which access modifier will be applied to it is getting by following table.

Base Class member Access Specifier	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not Accessible	Not Accessible	Not Accessible

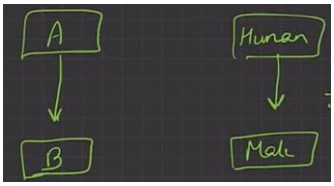
Here, base class means super class.

From this chart we can conclude that private data member of super class can't be inherited.

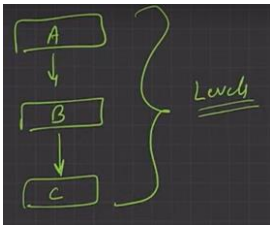
Protected: It's similar to private modifier. You can't access the data outside of that class. But here's a condition that its child classes can access data members of it.

Types of Inheritance:

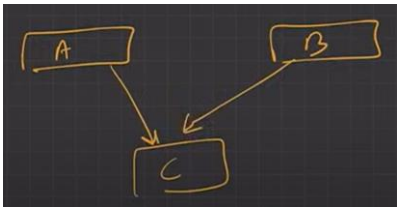
1. Single: Inheritance from one class to another.



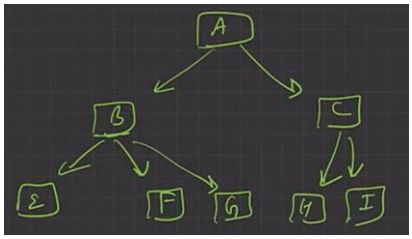
2. Multi-level: Inheritance in multiple levels.



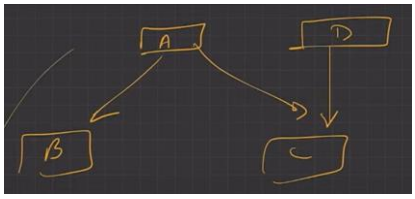
3. Multiple



4. Hierarchical: One class serve as parent class for more than 1 class.



5. Hybrid: Combination of more than one type of inheritance.



Inheritance Ambiguity:

If a class C is inherited from parent class A and B and both contain a function with same name then when we call that function with object created from C, it will get confused to which function that call is.

To solve this problem, we use scope resolution operator (::).

There's a huge difference between OOPs that we're learning and OOPs that we actually used in real industry. Concepts are same but there things are more based on the design patterns.

In real life, OOPs is used extensively because we've to reuse the code, make it readable and manageable.

Polymorphism:

Poly means many and morph means forms, it means multiple forms.

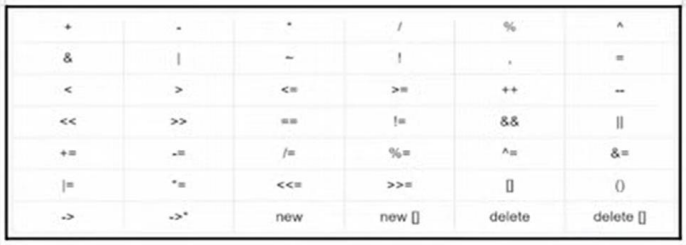
Ex – Your father is father for you, husband for wife, son for grandfather .... Etc.

There're two types of polymorphism:

1. Compile time polymorphism: (Static Polymorphism)
   We know what things exist in which forms during compile time before the run time.
   a. Function overloading:
      If you want to use same function multiple times the you can define it by changing parameters in each function definition. (changing function signature)
      If you only change the return type of function then it will not work. If you want to overload then must change the input parameters.

```
void sayHello() {
    cout << "Hello unknown user!" << endl;
}

Codiumate: Options | Test this method
void sayHello(string name) {
    cout << "Hello " << name << "!" << endl;
}
```

b. Operator overloading:
   You can able to overload following operators,



+	-	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	--
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

   But you can't able to overload (::, ?:, ., .*) operators.

   Overloading means changing it's definition. Ex – after + operation it's doing – operation etc.

2. Run time polymorphism (Dynamic Polymorphism)
   If something is exist in multiple forms, but you don't know it at compile time, it will know only at runtime.
   a. Method Overriding:
      Method name of parent and child class must be same.
      Method parameter of parent and child class must be same.
      Only possible through inheritance.

Abstraction: (Implementation hiding)

Only show essential things and not all.

Ex – If you're sending email then you only need to write it and send it by clicking button. You don't need to know about its implementation, what protocol is used....etc.

More to learn:

Learn OOPs in C++ from codewithharry's playlist → click here

Encapsulation VS abstraction

Interfaces

Friend function