Day:

Date: / /

Inheritance.

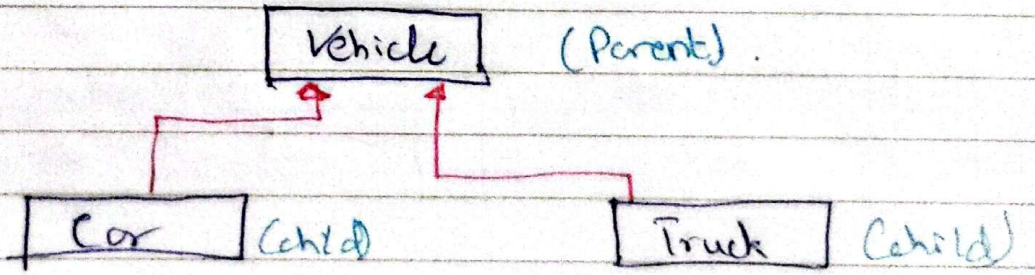
Inheritance allows a new class to be based on an existing class. The new class automatically inherits all the member variables and functions. except Constructor and destructor. of the class it is based on.

- It can be said to be a parent-child relationship.
- it is "is a" relationship
- In UML's it is represented by:

"child ⟶ parent"

Example:

- a car is a vehicle.
- a truck is a vehicle.

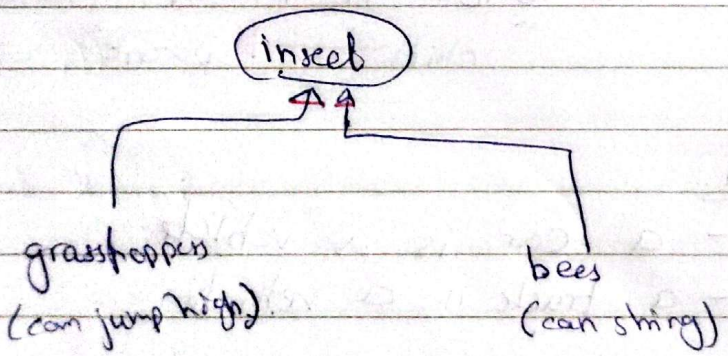


Yousaf

Date: / /

The concept of generalization and specialization:

- In real world we can find many objects that are speciallized version of other more general objects
- Example ~~teste~~ a the term insect describles a very general ~~term~~ types of creatures with numerous same. characteristics.
  So Grass Hoppes ~~am~~ & Bees are insects because they have all the general characteristics of insects. But Furthermore, they have some speeical abilities i.e grass hoppes can jump high ~~bob~~ and bees can sting.

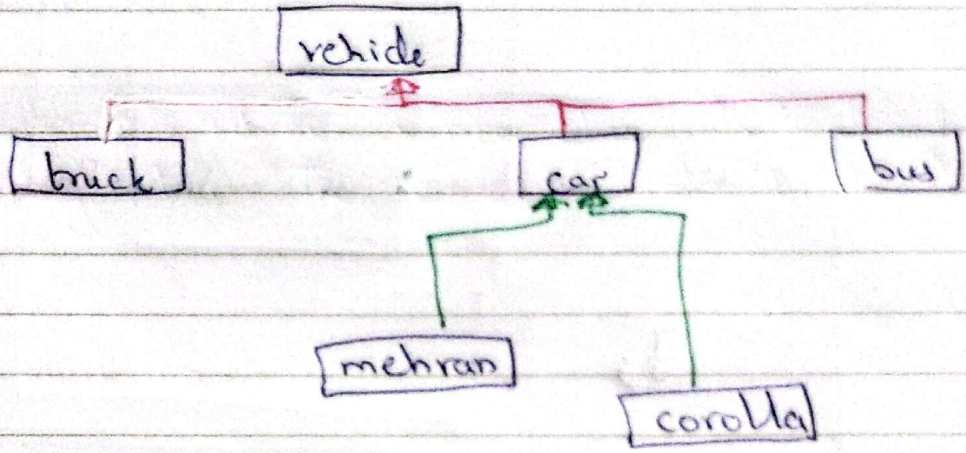


Yousaf

Day: Date: / /

More example of inheritance:

1)



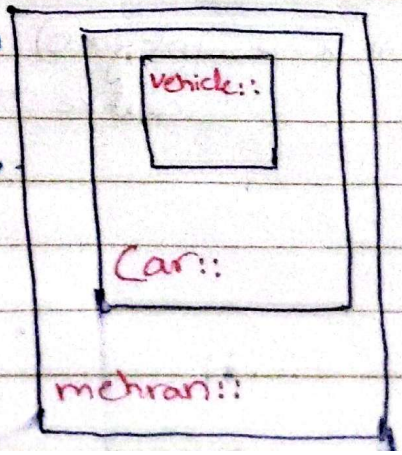
- car, bus and truck inherit from vehicle.
- mehran and corolla inherit from car.

= car will have all the ~~characters~~ member variables/~~functions~~ of vehicle.

= mehran will have all the member variables/function of car & vehicle.
(bcz car already have all veribles/functions of vehicle).

an easy way to interpret mehran:

= Mehran contains Car which contains vehicle.



- "::", these are globed scope operator, written to show scope of all classes.

Yousaf

Date: / /

How to do inheritance in cpp?:

```
class ClassName : access modifier Base/Parent class name {
    . . . .
    . . . . .
};
```

- access modifiers = Public, Private, protected.

Example:

```
1  class A{
2  private:
3      int x, y;
4
5  public:
6      void display (){
7                  cout << x << " " << y << endl;
8              }
9      A( int x=0, int y=0) : x(x), y(y) { cout << " conc A " << endl; }
10
11
12 };
13
14
```

Yousaf

Day: Date: / /

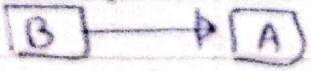
```
15
16
17  class B : public A {
18
19  public:
20          B( int x=0 , int y=0) : A(x, y) {
21              cout<<"conc B"<<endl;
22          }
23  };
24
25  int main(){
26
27      B b(2,3);
28
29      b.display() //also b.A::display() ✓allowed
30  }
```

Terminal:

```
conc A
conc B
2  3
```

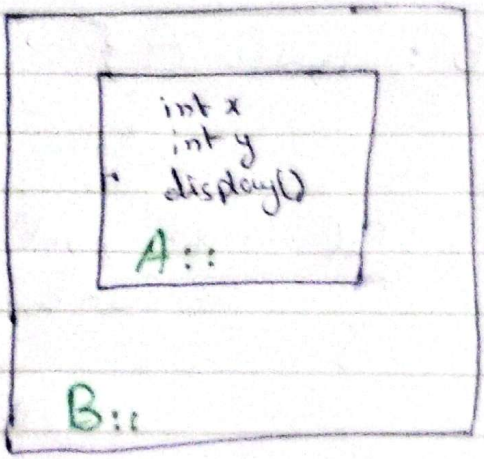
What is happening in code:

- class B is a child of class A.



so B inherits everything from A.

Yousaf



Figure

- When B is created in main, Firstly the Parents are made i.e A. This is why A's constructor is called first.

Line 29:

- Here b object is calling a function made in A. This is totally possible. Why? See the figure on top of page:

  we can see that A lies inside B so B can access A's display.

Extra:

- But Here is a problem!!
  - we cannot access x and y inside B bez they are private for A. How to Fix → next page.

Yousaf

Day: Date: / /

Protected Members:

- Like public and private, protected class access also exists.

Protected members of base class are like private members but they CAN be accessed by derived classes. The base class access specifications determines how private, public and protected base class members are accessed when they are inherited by base classes.

Private	Only inside the Base class. Derived class, Not allowed to use.
Protected	Inside BASE & derived classes. Banned in global or any other place.
Public	Available everywhere.

Yousaf

Day: Date: / /

Example:

```
Class A {

protected:
        int x;
        int y;

  void display() { cout ---- ---- }.

};

class B: public A {

public:
        void Change ( int x1, int y1) {
                x = x1;
        }       y = y1;

};
```

Yousaf

Date: / /

```
int main (){
    B b;
    b.change (5,3);
    b.display();
}
```

✓ allowed

✗ Error

What's Happening?

- change function is allowed bcz x and y are now freely available to be used in B class.

- Display function is NOT allowed bcz it is a protected member of class A, it is allowed in B but not in main or anywhere else.

Fix:
make display function public.

Yousaf

Date: / /

Class access:

When we inherit classes we write public, private or protected before them, this can completly alter the class acess.

class B : access modifier A {...};

This can alter class access.

Base class (original)	Access Modifier applied.		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	unaccessible	unaccessible	unaccessible

Table explanation:

The table shows what the original access was of members and then how it changes based on the modifier applied.

Yousaf

Day: Date: / /

Example of how public changes to protected:

```
class A {

  public:
      void display () { cout << "Haha" << endl; }

};

class B : protected A { };

int main () {
    A a;
    B b;
    b.display (); ✗ error.
    a.display (); ✓ allowed.
}
```

Now if we access display from B., display has been made protected and so cannot be accessed in global or other places.

- This is only if we access member of A THROUGH B.

Yousaf

Page Date: / /

- also if extend the chain and say:

class C : public B { };

even now display is allowed inside this class BUT not outside

- as we can see in the table if we ~~can~~ try to change protected to public, it will remain protected.

Example of ~~protected~~ private:

- consider the same protected one example but change protected to private. also consider Class C to exist.

- Now clear C CANNOT access display also.

why?:

In class B, display became private so now we cannot access it ~~directly~~ in C+ like we saw how we could not use x, y in when they were private in the section before protected members.

Yousaf

Day: Date: / /

Constructors and destructors in base and derived classes:

> The base class's constructor is called Before derived class's constructor.

> The destructors are called in reverse so Derived /child class's destructor is called first.

Be careful while using member initializer.

Consider int x, y to be variables of a class A which inherits to B.

```
class B: public A{

Public:
    B(int x1, int y1) : x(x1), y(y1) { }.
```

✗ ✗

Error.

x and y haven't even been made yet. member initializer is at very start of a program.

Fix: (2 ways):

1) `B(int x1, int y1) : A(x1, y1) { }.`
2) `B(int x1, int y1) { x=x1; y=y1; }`

Yousaf

Day: Date: / /

- in 1) way we are using the parametrized constructor of A. if it exist. At member intiallizer can parent object is being made so we can use it's constructor to makee it.

- In 2) way objects have cetreaely been made so now x and y are veriables of B bo and we can assign value.

Imp Remember:

if Class A had eg x as an CONST int. then we ~~will~~ can ONLY follow first way of member intiallizer as other way is likee changing const value which is not possible.

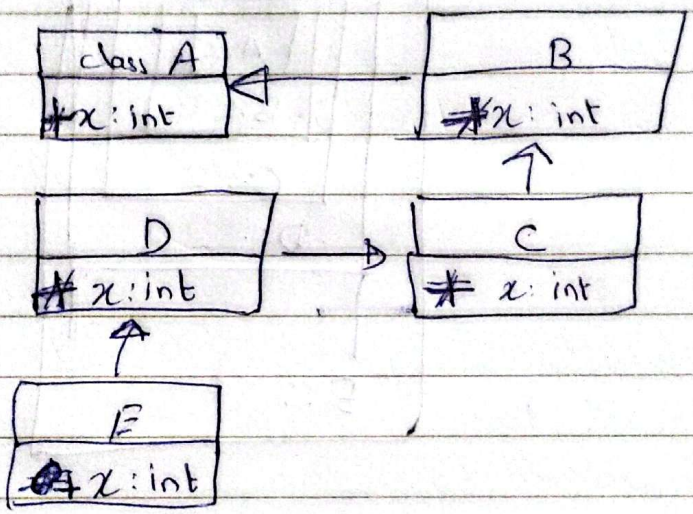
Yousaf

Day: Date: / /

Class Hierarchies:

- There can be a very long chain of inheritances.

But what if they all had a same member variable x.

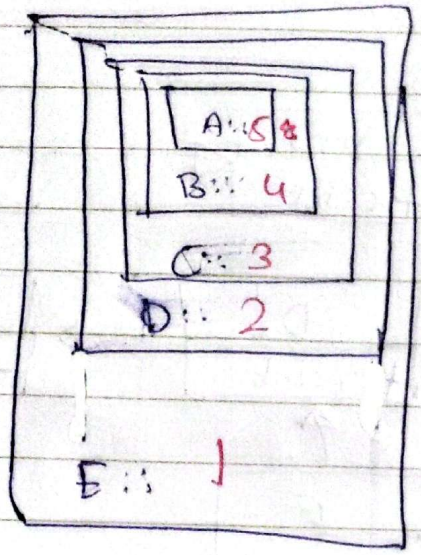



```
int main() {

    E e;

    cout << e.x << endl;

}
```

The output will be the x of class E. The most childest or youngest class.

Day: Date: / /

- This shows that the most newest or youngest class has most priority.

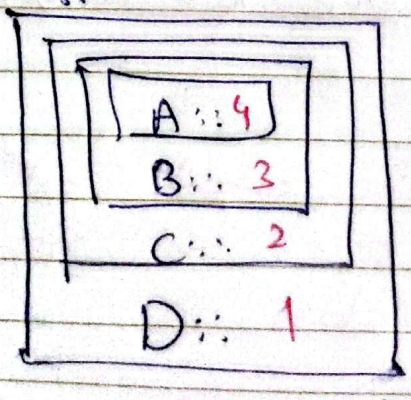


Priorities
1 → 5
decreasing.

What if I wrote:

```
D d;
cout<<d.x<<endl;
```

This means:



so this will output 4 x of d.

Yousaf

Date: / /

So: How to access other x's:

we can use a simple concept of Scope resolution operator "::";

cout << e.A::x << endl; ✓

cout << e.C::x << endl; ✓ works.

.

cout << d.B::x << endy. ✓

This allows us to use those particular members.

// This concept is not only for variables BUT all Members like functions.

Pg: Date: / /

Redifined ~~Overridden~~ Functions / Shadowing:

This is having the same name member of function in both base and derived classes.

- This is the same concept we started in last section.

- The Highest Priority is of the most childest or the outermost class i make in diagrams.

- The members of other BASE Classes CAN ONLY BE USED if:
  1) No ~~overridden~~ redifined function in derived class
  2) using scope operator: "::"

Instead of these 2 conditions there is no way of using them.

Page: Date: / /

Example:

```
class A {
public:
    void print (string x){
        cout << "In A" << endl;
    }
};

class B : public A {
public:
    void print (int x){
        cout << "In B" << endl;
    }
};

int main (){
    B b;

    b.print (5); ✓ allowed
    // b.print ("Hi"); X error.
    b.A::print ("Hi"); ✓ allowed.
}
```

Pg:

Date: / /

why doesn't b.print("Hi") work?

when we use this function, b call it's own print but it has a parameter as an int which CANNOT be converted into string so its was an error.

Also A's print was NOT USED bcz priority is of b's print.

Fix::

1) use A::print as shown in 4th line of main.
2) change names of function and use each function with its respective name.
3) remove the print function from B and function of A will be used.

// In this way we were able to differentiate overloaded and redifined functions.

Yousaf