المحاضرة التاسعة

الوراثــة

Inheritance

مفهوم الوراثة في (**oop**) يعني إمكانية وراثة صنف ما لمواصفات وخصائص صنف آخر وبناءاً على ذلك فإنه يمكن تحديد مفهومين جديدين هما :

1. صنف القاعدة (الأساس) Base Class

وهو الصنف الذي يحوي البيانات و الدوال المراد توريثها إلى صنف آخر ..

[الصنف الموروث من قبل صنف آخر]

2. الصنف المشتق Derived Class

وهو الصنف الوارث لصفات صنف القاعدة .

☑	عندما يرث الصنف المشتق صنف القاعدة فإن كل خصائص ووظائف صنف القاعدة تكون متاحة للصنف المشتق إضافة لذلك يمكن للصنف المشتق تعريف خصائص ووظائف جديدة .

أنواع الوراثة:

نوع الوراثة يحدد درجة الوصول للبيانات (المتغيرات) والوظائف (الدوال) في الصنف المشتق .

وتوجد ثلاثة أنواع :

1. الوراثة الخاصة Private inheritance

عندما يرث صنف مشتق صنف القاعدة وكان نوع الوراثة "خاص **Private** " فإن كل خصائص ووظائف صنف القاعدة ستصبح خاصة في الصنف المشتق .

2. الوارثة المحمية Protected inheritance

في هذا النوع من الوراثة ستصبح بيانات ووظائف صنف القاعدة محمية في الصنف المشتق .

3. الوراثة العامة Public inheritance

عندما يكون نوع الوراثة عام فإنه سوف يتم توزيع البيانات و الوظائف كالآتي :-

أ. المحمي في صنف القاعدة يصبح محمي في الصنف المشتق

ب. العام في صنف القاعدة يصبح عام في الصنف المشتق .

جدول يوضح تفاصيل الوراثة :

نوع الوراثة / نوع بيانات صنف القاعدة	Private	Protected	Public
Private	لا يورث	لا يورث	لا يورث
Protected	Private	Protected	Protected
Public	Private	Protected	Public

☑	النوع **Private** لا يورث أبداً بينما الأنواع التي تورث هي **Protected** و **Public** فقط .

الصيغة العامة للوراثة

القاعدة

```
class        base
{
    ................
    ................
};
```

المشتق

```
class        derived
{
    ................
    ................
};
```

ولجعل الصنف المشتق **derived** يرث من صنف القاعدة **base**

نكتب الآتي :-

```
class   derived    : < inheritance type > :base
                     نوع الوراثة
{
.........................................
.........................................
.........................................
};
```

مثال :-

```
    // base class
  class     B
  {
        int  i ;
     public :
        set _ i ( int     n ) ;
        int   get _ i (    ) ;
    };
    // derived class
class  D : Public  B
    {
         int   j ;
      public :
        set _ j ( int   n ) ;
        int   mul (      ) ;
};
```

class D : Public B تعنى أن الصنف D يرث جميع صفات الصنف الموروث B	☑

والآن نكتب البرنامج بصورة متكاملة

```
#include<iostream.h >
// base class
class base
{
  int  i;
  public:
   set_i( int n ) ;
   int get_i( ) ;
} ;
// derived class
class derived:public base
{
   int j;
   public:
    set_j(int n) ;
    int mul( );
};
// value of I in base class
base ::set_i(int n)
{
i=n;
}
// return value of i to base class
```

```
int  base::get_i( )
   {
     return i ;
   }
   // value of j in derived class
derived::set_j(int n)
{
 j = n ;
// return i from base class and j
// from drived class
}
int derived::mul( )
{
  return j * get_i( );
  }
int main ( )
{
 derived ob;
 int i,j;
 cout<< "enter i and j \n ";
 cin>>i>>j;
 ob.set_i(i);
 ob.set_j(j);
 cout<<ob.mul( );
return 0;
}
```

أعضاء الصنف المحمية : Protected Members of class

كما هو المعروف من مبدأ التوارث بأن الصنف الوارث **derived class** لا يستطيع أن يصل إلى الأعضاء الخاصة **Private Members** في الصنف الموروث **bass class** .

ولكن قد تظهر الحاجة في بعض البرامج إلى ضرورة استخدام الأعضاء الخاصة في الصنف الموروث مع إبقاءها خاصة ولهذا يمكن استخدام الكلمة المحجوزة **Protected**

أن **Protected** مكافئ (شبيه تماماً) بالكلمة المحجوزة **Private** ولكن **Protected** تسمح لجميع الأصناف الوارثة من استخدام الأعضاء الخاصة في الصنف الموروث .

☑	1. الأعضاء المحمية **Protected** يمكن أن تقع في أي مكان داخل الصنف ولكن عادة تقع بين الأعضاء الخاصة و العامة . 2. إذا قام الصنف الوارث لوراثة أعضاء محمية من الصنف الموروث فأنها أتوماتيكياً تصبح أعضاء محمية داخل الصنف الوارث ..

مثال على الإعضاء المحمية :

```
#include<iostream.h>
class  samp
{
  private :
    int a ;
  protected :
    int b ;
  public :
    int c ;
samp ( int n ,  int  m )
{
a = n ;
b = m ;
}
int  geta(  )
{
 return a ;
}
int getb(   )
{
return b ;
}
};
int main ( )
{
 samp ob(10,20) ;
 ob.c = 30 ;
 cout<<ob.geta( )<<"\n" ;
cout<<ob.getb( )<<"\n" ;
cout<<ob.c<<"\n" ;
return 0;
}
```

☑	الأعضاء المحمية في الصنف تعتبر مغلقة (لا يمكن الوصول إليها) في أي صنف غير وارث ..

```
#include<iostream.h>
class    base
{
protected :
int a,b;
public:
void set_ab( int n,int m )
{
```

```
a = n;
b = m;
}
};
class derived:public base
{
 int c ;
public :
void set_c( int   n )
{
 c = n;
}
void show_abc( )
{
cout << a << " \t " << b << "\t " <<c<<"\n";
}
};
int main ( )
{
int a1,b1,c1;
cin>>a1>>b1>>c1;
derived ob;
ob.set_ab(a1,b1);
ob.set_c(c1);
ob.show_abc( );
return 0;
}
```

دوال البناء و الهدم تحت الوراثة :

Constructor and Destructor Functions with Inheritance

إن كلاً من الصنف الموروث **base class** والصنف الوارث **derived class** يمكن أن تمتلكان دوال بناء أو دوال هدم

☑	1. دالة بناء الصنف الموروث يتم تنفيذها قبل دالة بناء الصنف الوارث 2. دالة هدم الصنف الوارث تنفذ قبل دالة الهدم للصنف الموروث . وذلك يعود لاعتماد الصنف الوراث دائماً على الصنف الموروث واستقلالية الصنف الموروث عن الصنف الوارث .

مثال :

```
#include<iostream.h>
class base
{
public:
base ( )
{
cout<<"Constoructor of base class \n";
   }
~base( )
{
cout<<"Destructar of base class \n";
}
};
class derived:public base
{
public:
derived ( )
{
cout <<"Constructor of derived class \n";
}
~derived ( )
{
cout<<"Destructer of derived class \n";
}
};
int main ( )
{
derived  ob;
return 0;
}
```

Out Put :-

```
Constructor of base class
Constructor of derived class
Destructor of derived class
Destructor of base class
```

تبادل المعاملات مع دالة بناء الصنف الوراث :-

تتم بطريقة اعتيادية ويمكن توضيحها من خلال المثال التالي :

مثال :

```
#include<iostream.h>
class base
{
int i;
public :
base( int n )
   {
cout <<" Costructor of base class \n";
   i = n;
}
~base( )
{
cout <<"Destructor of base class \n";
}
showi( )
{
cout<<i<<"\n";
   }
   };
class dervied : public base
{
int  j;
public:
dervied (int n):base (n)
//transfer perrameter to base class
{
cout <<"canstrueter of derived class \n";
 j = n;
}
~dervied ( )
{
cout <<"destrueter of derived class \n";
}
showj( )
{
cout << j <<"\n";
}
};
```

```
int main ( )
{
int a;
cout <<" enter a \n ";
cin >> a;
dervied ob(a);
ob.showi( );
ob.showj( );
return 0;
}
```

☑	في البرنامج أعلاه دالة بناء الصنف الموروث و الوارث تمتلك نفس المعامل ويقوم الصنف الوارث بإرسال هذا المعامل إلى الصنف الموروث . حيث أن المعامل **n** يتم إرساله من الصنف الوارث إلى الموروث من خلال العبارة : **derived (int n):base (n)**

مثال :- يوضح كيفية إرسال معامل معين إلى دالة بناء الصنف الوارث وآخر إلى دالة الصنف الموروث

```
#include<iostream.h>
class   base
{
int i;
public :
base( int  n )
{
cout<<"costructor of base class \n ";
i=n;
}
~base ( )
{
cout<<"destructor of base class \n";
}
showi( )
{
cout<<i<< "\n";
}
};
class derived:public base
{
int j;
public:
derived ( int  n , int m ):base ( m )
//transfer perrameter to base class
```

```
{
cout <<"destructor of base class \n";
 j = n;
}
showj( )
{
cout << j << "\n";
}
};
int main ( )
{
int a,b;
cout<< "enter a and b \n";
cin>>a>>b;
derived ob (a,b);
ob.showi( );
ob.showj( );
return 0;
}
```