

كلية الهندسة – قسم المعلوماتية
مقرر برمجة 2

أ. د. علي عمران سليمان

محاضرات الأسبوع السادس

inheritance

الفصل الاول 2023-2024

1. Introduction.
 2. Defining inheritance.
 3. Create an inheritance relationship between the base class and the derived class.
 4. Identify the relationship between inheritance and protected class members.
 5. Identify the different forms of inheritance (public, private, protected).² Create multiple inheritance.
 6. Knowing the relationship between inheritance and the functions of construction and demolition.
 7. Inheriting virtual classes.
1. مقدمة.
 2. تعريف الوراثة
 3. إنشاء علاقة وراثة بين صنف أساس وصنف مشتق.
 4. التعرف على العلاقة بين الوراثة والأعضاء المحمية للصنف.
 5. التعرف على الأشكال المختلفة للوراثة (عامة، خاصة، محمية).
 6. إنشاء الوراثة المتعددة.
 7. معرفة العلاقة بين الوراثة وتوابع البناء والهدم.
 8. وراثة الأصناف الظاهرية.

المحاضرة من المراجع :

[1]- Deitel & Deitel, C++ How to Program, Pearson; 10th Edition (February 29, 2016)

[2]- د.علي سليمان, البرمجة غرضية التوجه في لغة C++ 2009-2010

- إنشاء صنف عام يقوم بتعريف وصف مشترك لمجموعة من العناصر المترابطة.
- وراثه الصنف من قبل أصناف أخرى أكثر تخصيصاً تتضمن أعضاء أخرى إضافة إلى الأعضاء المورثة.
- ندرس مفهوم البرمجة غرضية التوجه وما ينطوي عليه من تطوير في مفهوم البرمجة ومنها:

✓ التغليف **encapsulation**: أي الربط بين الشيفرة والبيانات التي تتعامل معها بشكل يؤمن حمايتها من التدخل الخارجي أو سوء الاستخدام تمت دراسته.

✓ تعددية الأشكال **polymorphism**: أي إمكانية قيام واجهة واحدة بالتحكم بالوصول إلى مجموعة من الوظائف .

✓ الوراثة **inheritance**: أي إمكانية حصول غرض على خصائص غرض آخر.

- نتوقف في بحثنا الحالي عند النقطة الأخيرة التي تتيح الاستفادة من خاصية إعادة استخدام البرمجيات بفعالية عالية.

• لفهم هذا الأمر نحاول طرحه في هذه الفقرة من خلال الأمثلة التالية:

المثال الأول:

- نرغب بتطوير برنامج يتعامل مع الأشكال الهندسية ويستخدم من بين هذه الأشكال الهندسية الشكلين التاليين:
 - ✓ الأول هو النقطة point والمعرفة في جملة الإحداثيات الديكارتية الثنائية من خلال إحداثياتها على المحورين X و Y،
 - ✓ الثاني هو الدائرة circle وهي معرفة في جملة الإحداثيات الديكارتية الثنائية من خلال مركزها وهو عبارة عن نقطة (معرفة من خلال إحداثياتها على المحورين X و Y) ومن خلال نصف قطرها.

- إن البرنامج الواجب بناؤه يتضمن تعريف صنفين:
- الأول لتعريف النقطة:

// Point class definition represents an x-y coordinate pair.

```
class Point {  
public:  
    Point(int xValue=0,int yValue=0) // default constructor  
    {  
        x = xValue;  
        y = yValue;  
    } // end Point constructor
```

Introduction

```
void setX( int xValue ) // set x in coordinate pair
{   x = xValue;   } // end function setX

int getX() const // return x from coordinate pair
{   return x;   } // end function getX

void setY( int yValue ) // set y in coordinate pair
{   y = yValue;   } // end function setY

int getY() const // return y from coordinate pair
{   return y;   } // end function getY

void print() const // output Point object
{   cout << '[' << x << ", " << y << ']' ; } // end function print

private:   int x; // x part of coordinate pair
           int y; // y part of coordinate pair
}; // end class Point
```

Introduction



مقدمة

الثاني لتعريف الدائرة:

```
class Circle {
public:
    Circle(int xValue=0,int yValue=0,double radiusValue=0.0)
    {   x = xValue;   y = yValue;   setRadius( radiusValue );
    }
    // default constructor
    // end Circle constructor

    void setX( int xValue ) // set x in coordinate pair
    {   x = xValue;   } // end function setX

    int getX() const // return x from coordinate pair
    {   return x;   } // end function getX

    void setY( int yValue ) // set y in coordinate pair
    {   y = yValue;   } // end function setY
}
```

Introduction



مقدمة

```
int getY() const // return y from coordinate pair
{ return y; } // end function getY
```

```
void setRadius( double radiusValue ) // set radius
{ radius = ( radiusValue < 0.0 ? 0.0: radiusValue );
} // end function setRadius
```

```
double getRadius() const // return radius
{ return radius; } // end function getRadius
```

```
double getDiameter() const // return diameter
{ return 2 * radius; } // end function getDiameter
// calculate and return circumference
```

```
double getCircumference() const // return circumference
{return 3.14159*getDiameter();} // end function getCircumference

// calculate and return area
double getArea() const // return area
{ return 3.14159 * radius * radius; } // end function getArea

// output Circle object
void print() const // output Circle object
{ cout << "Center = [" << x << ", " << y << ']' << "; Radius = "
  << radius; } // end function print

private: int x; // x-coordinate of Circle's center
         int y; // y-coordinate of Circle's center
         double radius; // Circle's radius
}; // end class Circle
```

هنا يوجد الكثير من التكرار في الكود البرمجي.

Rectangle

المستطيل

المثال الثاني:

- يتعامل هذا البرنامج مع نوعين من الأشكال الهندسية هما المستطيل `rectangle` والمعرف من خلال طوله `length` وعرضه `width` ومتوازي المستطيلات المعرف من خلال طول قاعدته `length` وعرضها `width` وارتفاعه `height` .
- يجب أن يتضمن البرنامج الواجب بناؤه تعريف صنفين:

```
// Rectangle class definition.
```

```
class Rectangle {
```

```
public:
```

```
    Rectangle(double L=0.0, double W=0.0) // default constructor
```

```
    { length = L; width = W; } // end Rectangle constructor
```

```
    void setLength( double L ) // set length
```

```
    { length = L; } // end function setLength
```

Rectangle

```
double getLength() const           // return length
{   return length;   }           // end function getLength

void setWidth( double W )         // set width
{   width = W;   }           // end function setWidth

double getWidth() const          // return width
{   return width;   }           // end function getWidth

double getArea()                 // return Area
{   return length*width;   } //end function getArea

void print() const               // output Rectangle object
{   cout << "Length= " << length << "\t Width= " << width
<< endl;   }           // end function print

private: double length; double width;}; // end class Rectangle
```

Cuboid



متوازي المستطيلات

الثاني لتعريف متوازي مستطيلات:

```
// Rect class definition.
class Cuboid {
public:
    Cuboid (double L=0.0, double W=0.0, double H=0.0)
    {length= L; width = W; height=H;} // end Cuboid constructor

    void setLength( double L ) // set length
    {length = L; } // end function setLength

    double getLength() const // return length
    { return length; } // end function getLength

    void setWidth( double W ) // set width
    { width = W; } // end function setWidth
```

Cuboid



متوازي المستطيلات

```
double getWidth() const // return width
{ return width; } // end function getWidth

void setHeight( double H ) // set height
{ height = H; } // end function setHeight

double getHeight() const // return height
{ return height; } // end function getHeight

double getArea() // return Area
{ Return 2*((length*width)+(length*height) //end function getArea
  +(width*height)); }

double getVolume() // return volume
{Return length*width*height; } //end function getVolume
```

Cuboid

```
void print() const           // output Cuboid object
{
    cout << "Length= " << length;
    cout << "\t Width= " << width;
    cout << "\t Height= " << height << endl;
}
// end function print

private:
    double length;
    double width;
    double height;
};
// end class Cuboid
```

- إن هذا التكرار في إعادة كتابته للتوابع ضمن المثالين السابقين لا يعد من الممارسات البرمجية الجيدة من وجهة نظر هندسة البرمجيات.
- قدمت البرمجة غرضية التوجه طريقة لتجنب مثل هذا الأمر من خلال مفهوم الوراثة.

3-2- تعريف الوراثة

Defining inheritance

- تعتبر الوراثة أحد أشكال إعادة استخدام النصوص البرمجية حيث يتم إنشاء أصناف جديدة انطلاقاً من أصناف موجودة مسبقاً.
- تأخذ الأصناف الوارثة صفات وتصرفات الأصناف المورثة وتضيف عليها إمكانيات جديدة تحتاجها أو التعديل على ماتمت وراثته دون التأثير على الصنف المورث.
- تساعد على توفير الجهد والزمن والتكلفة اللازم لتطويرها وتشجع استخدام برمجيات موثوقة وفعالة.
- تسمى الأصناف الموجودة مسبقاً بالأصناف الأساس (Base, Parent, Super) classes وتسمى الأصناف الوارثة عنها بالأصناف المشتقة (Derived, Child, Sub) classes.
- يشترك صنف من صنف أساس من خلال **الشكل العام** التالي لتعريف الوراثة:

```
class derived_class_name : access baseClassName  
{ /* body of class */  
};
```

- عندما يرث صنف صنف آخر، فإن أعضاء الصنف الأساس تصبح أعضاء في الصنف المشتق (حسب نوع الوراثة). والعكس غير صحيح أي أن أعضاء الصنف المشتق لا تصبح أعضاء في الصنف الأساس.

Defining inheritance

- يحدد المحدد access نوع الوصول أو نوع الوراثة إلى أعضاء الصنف الاساس من قبل الصنف المشتق. حيث تكون الوراثة بحسب نوع access إما خاصة private، عامة public أو محمية protected.
- فيما يلي نناقش هذه الأنواع الثلاثة، ولكن قبل ذلك لنلق نظرة على محدد الوصول protected .
- تعرفنا لدى دراسة الأصناف في الفصول السابقة على محدد الوصول private و public حيث قلنا أن هذين المحددين يحددان إمكانية الوصول إلى الأعضاء المعرفة ضمنهما من قبل باقي أجزاء البرنامج، ونقول الأعضاء العامة متاحة الوصول (كثيراً) والخاصة ضيقة الوصول ولاتورث.
- يوفر المحدد protected لدى استخدامه ضمن تعريف الصنف في توفير مستوى وسيط بين مستوى الوصول العام ومستوى الوصول الخاص، يحميها ضمن الصنف مثل الخاص ويورثها للصنف المشتق مثل العام.

Public inheritance

- لتعريف الوراثة العامة تكون قيمة المحدد `access` الوارد في الصيغة العامة السابقة هي `public` وعندها فإن جميع الأعضاء العامة في الصنف الأساس تصبح أعضاء عامة في الصنف المشتق وجميع الأعضاء المحمية في الصنف الأساس تصبح أعضاء محمية في الصنف المشتق أما الأعضاء الخاصة للصنف الأساس فإنها تبقى خاصة بالنسبة له ولا تصبح أعضاء في الصنف المشتق (لاتورث).
- مثلاً: نقوم بإعادة تعريف الصنفين `Point` و `Circle` من المثال في الفقرة السابقة نعرف الصنف `Point` بشكل طبيعي كصنف أساس ثم نعرف الصنف `Circle` كصنف مشتق من الصنف `Point` (أي أن `Circle` هو `Point` ولكن لديها نصف قطر).

أولاً: الملف الرأسي لتعريف الصنف `Point` (الملف `Point.h`)

```
#ifndef POINT_H
#define POINT_H
class Point {
```


Class Point.h

public:

```
Point( int = 0, int = 0 ); // default constructor
void setX( int );          // set x in coordinate pair
int  getX() const;        // return x from coordinate pair
void setY( int );          // set y in coordinate pair
int  getY() const;        // return y from coordinate pair
void print() const;       // output Point object
```

private:

```
int x; // x part of coordinate pair
int y; // y part of coordinate pair
};     // end class Point
#endif
```

PointF.cpp

ثانياً: ملف تعريف توابع الصف PointF (الملف PointF.cpp)

```
#include "stdafx.h"
#include <iostream>
#include "point.h" // Point class definition
using namespace std;
Point::Point( int xValue, int yValue ) // default constructor
{ x = xValue; y = yValue; } // end Point constructor
// set x in coordinate pair
void Point::setX( int xValue )
{ x = xValue; } // end function setX
// return x from coordinate pair
int Point::getX() const
{ return x; } // end function getX
```

PointF.cpp

```
// set y in coordinate pair
void Point::setY( int yValue )
{   y = yValue; } // end function setY

// return y from coordinate pair
int Point::getY() const
{   return y;} // end function getY

// output Point object
void Point::print() const
{
    cout << '[' << x << ", " << y << ']' ;
} // end function print
```

Circle.h

ثالثاً: الملف الرأسى لتعريف الصف Circle (الملف Circle.h)

```
#ifndef CIRCLE_H
#define CIRCLE_H
#include "point.h"
class Circle: public Point {
public:
    Circle( int = 0, int = 0, double = 0.0 ); // Point class definition
    void setRadius( double ); // default constructor
    double getRadius() const; // set radius
    double getDiameter() const; // return radius
    double getCircumference() const; // return diameter
    double getArea() const; // return circumference
    void print() const; // return area
    // output Circle object
```

CircleF



الوراثة العامة للصنف الأساس

```
private: double radius; // Circle's radius
}; // end class Circle
#endif
```

رابعاً: ملف تعريف توابع الصنف CircleF (الملف CircleF.cpp)

```
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include "circle.h" // Circle class definition
using namespace std;
// default constructor
Circle::Circle( int xValue, int yValue, double radiusValue )
{ x = xValue; y = yValue; setRadius( radiusValue );
} // end Circle constructor
```

CircleF



الوراثة العامة للصف الأساس

```
// set radius
void Circle::setRadius( double radiusValue )
{   radius = ( radiusValue < 0.0 ? 0.0: radiusValue );
} // end function setRadius

// return radius
double Circle::getRadius() const
{   return radius;} // end function getRadius

// calculate and return diameter
double Circle::getDiameter() const
{   return 2 * radius;} // end function getDiameter

// calculate and return circumference
double Circle::getCircumference() const
{   return 3.14159 * getDiameter(); // end function getCircumference
```

CircleF

```
// calculate and return area
double Circle::getArea() const
{   return 3.14159 * radius * radius;}    // end function getArea

// output Circle object
void Circle::print() const
{   cout << "Center = [" << x << ", " << y << ']'
        << "; Radius = " << radius;
}    // end function print
```

- الصف Circle لم يتضمن تعريف البيانات الأعضاء x ، y والتوابع الأعضاء getX() ، setX() ، getY() و setY() وذلك نظراً لأن الصف Circle قد قام بوراثة من الصف Point.
 - إن التعريف السابق يتضمن خطأ يتمثل في تابع البناء للصف Circle وذلك لأن هذا التابع يحاول الوصول إلى البيانات الأعضاء الخاصة للصف Point وهذا الأمر غير متاح.
- يمكن تجاوز هذا الخطأ بإحدى الطريقتين التاليتين:

- **الطريقة الأولى:** بتغيير تعريف الصنف Point بحيث توضع البيانات الأعضاء لهذا الصنف ضمن محدد الوصول protected. بحيث يصبح شكل تعريف الصنف Point كما يلي:

protected:

```
int x; // x part of coordinate pair  
int y; // y part of coordinate pair
```

- إن هذا التعديل لا يؤثر في عمل الصنف Point كما أنه لا يؤثر في تعريف باقي الملفات البرمجية السابقة ولا يؤثر في إمكانية قيام باقي أجزاء البرنامج بالوصول إلى البيانات الأعضاء المحمية إلا أن هذا الاستخدام للمعطيات المحمية يخلق مشكلتين وهما:
 - ✓ عدم استخدام التوابع الأعضاء للصنف المشتق لوضع قيمة ضمن المعطيات الأعضاء المحمية للصنف الأساس, بل يمكنه بكل سهولة تعديل قيمة هذه الأعضاء المرتبطة بالصنف المشتق والتي يجب كتابتها بشكل تكون فيه لها علاقة بتوابع الصنف الأساس (التوابع غير الخاصة) وليس على طريقة بناء وتطوير هذا الصنف.
 - ✓ يجب أن يقوم المبرمج بتعديل الصنف الأساس بكل حرية مع استمرار تقديم الخدمات إلى الاصناف المشتقة, وعند استخدام المعطيات المحمية وتعديل الصنف الأساس يجب إعادة بناء الصنف وفي المشتقة.

CircleF

الوراثة العامة للصف الأساس

الطريقة الثانية: بتغيير تعريف الصف Circle بحيث لا يتضمن وصولاً إلى البيانات الأعضاء الخاصة للصف Point وإنما يستخدم التوابع الموروثة للوصول إليها.

```
// default constructor
Circle::Circle(int xValue, int yValue, double radiusValue)
:Point(xValue,yValue)
{ setRadius( radiusValue );} // end Circle constructor

// output Circle object
void Circle::print() const
{cout<<"Center = ";Point::print(); //call class Point print()
  cout<< "; Radius = " << radius;} // end function print
```

عند تعريف تابع في الصف المشتق يملك نفس اسم التابع في الصف الأصل يجعله مخفي، يمكن ندائه من خلال إستخدام معامل تحديد المجال للتمييز بينهما.

يمكن اختبار الوراثة في هذه الحالة وعملية الوصول إلى البيانات الأعضاء المحمية من خلال برنامج الاختبار التالي:

CircleF

```
// inheritance150.cpp : main project file.
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include "Circle.h"
using namespace std;
int main()
{   Circle c( 37, 43, 2.5 ); // instantiate Circle object
    // display point coordinates
    cout<< "X coordinate is " << c.getX()<<"\nY coordinate is "
    << c.getY() << "\nRadius is " << c.getRadius();
    c.setX( 2 ); // set new x-coordinate
    c.setY( 2 ); // set new y-coordinate
    c.setRadius( 4.25 ); // set new radius
```

CircleF

```
// display new point value
cout << "\nThe new location and radius of circle are\n";
c.print();

// display floating-point values with 2 digits of precision
cout << setprecision( 2 );

// display Circle's diameter
cout << "\nDiameter is " << c.getDiameter();

// display Circle's circumference
cout<<"\nCircumference is "<< c.getCircumference();

// display Circle's area
cout << "\nArea is " << c.getArea();  cout << endl;
system("pause"); return 0;
} // end main
```

CircleF



الوراثة العامة للصف الأساس

الخرج:

X coordinate is 37

Y coordinate is 43

Radius is 2.5

The new location and radius of circle are

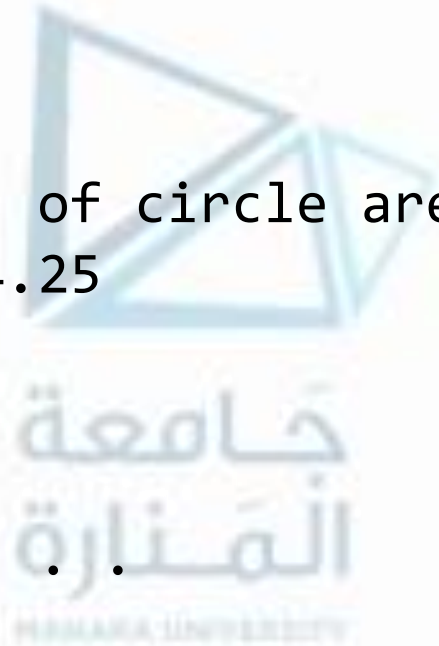
Center = [2, 2]; Radius = 4.25

Diameter is 8.5

Circumference is 27

Area is 57

Press any key to continue . . .



- الأعضاء الخاصة في الصنف الأساس لا يمكن الوصول إليها مهما كانت نوع الوراثة.
- لتعريف الوراثة الخاصة تكون قيمة المحدد `access` الوارد في الصيغة العامة السابقة هي `private` وعندها فإن جميع الأعضاء العامة والمحمية في الصنف الأساس تصبح أعضاء خاصة في الصنف المشتق.
- جعل الوراثة خاصة في المثال السابقة لا يصبح بالإمكان استخدام برنامج الاختبار السابق نظراً لأن التوابع `getX()` ، `setX()` .. وغيرها من التوابع المعرفة على أنها عامة ضمن الصنف `Point` ستصبح خاصة ضمن الصنف `Circle` وبالتالي لا يمكن الوصول إليها من خارج الصنف.
- لتعريف الوراثة المحمية تكون قيمة المحدد `access` الوارد في الصيغة العامة السابقة هي `protected` وعندها فإن جميع الأعضاء العامة والمحمية في الصنف الأساس تصبح أعضاء محمية في الصنف المشتق.
- يمكن أن نلخص الأنواع الثلاثة السابقة للوراثة من خلال الجدول التالي:

Private, Protected inheritance

الوراثة الخاصة, المحمية للصنف الأساس

النوع في المورث نوع الوراثة	private	protected	public
private	X	private	private
protected	X	protected	protected
public	X	protected	public

- ✓ الجزء الخاص في المورث مخبئة عن الصنف المشتق يمكن الوصول إليها مباشرةً من خلال التوابع الأعضاء غير الساكنة أو الصديقة باستخدام التوابع الأعضاء العامة أو المحمية المرتبطة بالصنف الأساس.
- ✓ وراثة خاصة: المحمي والعام في الأصل سيصبح خاص عند الوارث.
- ✓ وراثة محمية: المحمي والعام في الأصل سيصبح محمي عند الوارث.
- ✓ وراثة عامة: المحمي في الأصل سيظل محمي والعام سيظل عام عند الوارث.

أنواع الوراثة			محددات وصول أعضاء الصنف الأساسي
خاصة private	محمية protected	عامة public	
Private ضمن الصنف المشتق. يمكن الوصول إليها مباشرةً من خلال كافة التوابع الأعضاء غير الساكنة أو التوابع الصديقة.	Protected ضمن الصنف المشتق. يمكن الوصول إليها مباشرةً من خلال كافة التوابع الأعضاء غير الساكنة أو التوابع الصديقة.	public ضمن الصنف المشتق. يمكن الوصول إليها مباشرةً من خلال تابع عضو غير ساكنة أو تابع غير عضو أو صديق	Public
Private ضمن الصنف المشتق. يمكن الوصول إليها مباشرةً من خلال كافة التوابع الأعضاء غير الساكنة أو التوابع الصديقة.	Protected ضمن الصنف المشتق. يمكن الوصول إليها مباشرةً من خلال كافة التوابع الأعضاء غير الساكنة أو التوابع الصديقة.	Protected ضمن الصنف المشتق. يمكن الوصول إليها مباشرةً من خلال التوابع الأعضاء غير الساكنة أو الصديقة غير ثابت أو تابع غير عضو أو صديق	Protected
مخبئة عن الصنف المشتق يمكن الوصول إليها مباشرةً من خلال التوابع الأعضاء غير الساكنة أو الصديقة باستخدام التوابع الأعضاء العامة أو المحمية المرتبطة بالصنف الأساس	مخبئة عن الصنف المشتق يمكن الوصول إليها مباشرةً من خلال التوابع الأعضاء غير الساكنة أو الصديقة باستخدام التوابع الأعضاء العامة أو المحمية المرتبطة بالصنف الأساس	مخبئة عن الصنف المشتق يمكن الوصول إليها مباشرةً من خلال التوابع الأعضاء غير الساكنة أو الصديقة باستخدام التوابع الأعضاء العامة أو المحمية المرتبطة بالصنف الأساس	Private

Granting access

3-3- منح إمكانية الوصول

- عندما تتم وراثة صنف أساس وراثة خاصة، فإن جميع الأعضاء العامة والمحمية لذلك الصنف تصبح أعضاء خاصة للصنف المشتق.
- عند الحاجة للتعامل مع عضو أو أكثر وفق محدد الوصول الأصلي الخاص به (كما هو في الصنف الاساس).

• تحوي لغة C++ طريقتين لتحقيق ذلك:

1. باستخدام التعليمة `using`.

2. استخدام تصريح وصول ضمن الصنف المشتق.

يأخذ تصريح الوصول الشكل العام التالي:

```
base-class::member;
```

- يتم وضع تصريح الوصول تحت محدد الوصول الملائم في تصريح الصنف المشتق، نلاحظ أنه لا داعي لذكر النوع في تصريح الوصول.
- لمعرفة كيف يتم ذلك، لندرس حالة الوراثة الخاصة في المثال `Point/Circle` ونبينها فقط من خلال الملفين الرئيسيين للصنفين `Point` و `Circle`:

Granting access

- عند تعريف الوراثة خاصة لـ `Point` فيه توابع معرفة عامه تصبح خاصه ضمن الصنف الوارث ولا يمكن الوصول إليها من خارج الصنف ولنمنحها صفتها السابقة نضع أمامها اسم الصنف الأساس `Point::setX`.

```
class Circle: private Point {  
  
public:  
    Point::setX;    //makes setX public  
    Point::getX;   //makes getX public  
    Point::setY;   //makes setY public  
    Point::getY;   //makes getY public
```

- إننا نستطيع استخدام تصريح الوصول لاستعادة إمكان الوصول التي كان يتمتع بها العضو ضمن الصنف الأساس، ولكن لا يمكن استخدامه لرفع أو خفض مستوى الوصول إلى العضو في الصنف الأساس.
- إذا تم التصريح عن العضو على أنه خاص في الصنف الأساس، فلا يمكن جعله عاماً في الصنف المشتق.

Multi-level inheritance Hierarchy



3-4- الوراثة متعددة المستويات

- يمكن أن يتم إنشاء صنف مشتق إنطلاقاً من صنف مشتق آخر وفي هذه الحالة يجب فقط الانتباه إلى موضوع حالة التوابع والبيانات الأعضاء للصنف الأساس الأول ضمن الصنف المشتق الأول والذي أصبح أساساً للصنف الأساس الثاني وذلك بحسب نوع الوراثة في كل مستوى.
- نجد مثل هذا الاستخدام من خلال البنية **Point/Circle/Cylinder** حيث سيتم اشتقاق الصنف **Cylinder** من الصنف **Circle**.
- جرى التصريح عن المعطيات الأعضاء الخاصة ضمن الصنف **Point** وكذلك توابع الوضع **setx, sety** والقراءة **getx, gety** لها وتابع الطباعة **print**.
- أما الصنف **Circle** فقد تم اشتقاقه من الصنف **Point** وهو يتضمن الإمكانات التابعة للدائرة **getradius, getDiameter, getCircumference, getArea, print** وعند اشتقاق الصنف **Cylinder** يجب تعريف هذه التوابع مع ما يتناسب ووضعها وتحسب مساحة الدائرة وفق المعادلة πr^2 .
- تحسب مساحة الاسطوانة $(2 \pi r^2) + (2 \pi rh)$ وبالتالي يجب اعادة تعريف التابع **getArea** ضمن صنف **Cylinder**:

Cylinder.h

4-3- الوراثة متعددة المستويات

الملف الرأسى لتعريف الصنف Cylinder:

```
// Cylinder.h Cylinder class inherits from class Circle.
#ifndef CYLINDER_H
#define CYLINDER_H
#include "circle4.h" // Circle class definition
class Cylinder: public Circle {
public: // default constructor
    Cylinder(int =0, int =0, double =0.0, double =0.0);
    void setHeight( double ); // set Cylinder's height
    double getHeight() const; // return Cylinder's height
    double getArea() const; // return Cylinder's area
    double getVolume() const; // return Cylinder's volume
    void print() const; // output Cylinder
```

cylinderF.cpp

```
private:    double height; // Cylinder's height
}; // end class Cylinder
#endif
```

ملف تعريف الصنف CylinderF:

```
// cylinderF.cpp Cylinder class inherits from class Circle.
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include "Circle.h" // Circle class definition
using namespace std;
// default constructor
Cylinder::Cylinder(int xValue, int yValue, double radiusValue, double
heightValue):Circle( xValue, yValue, radiusValue )
{    setHeight( heightValue );} // end Cylinder constructor
```

cylinderF.cpp

```
// set Cylinder's height
void Cylinder::setHeight( double heightValue )
{height=(heightValue<0.0 ? 0.0:heightValue);}//end function setHeight

// get Cylinder's height
double Cylinder::getHeight() const
{ return height;} // end function getHeight

// redefine Circle4 function getArea to calculate Cylinder area
double Cylinder::getArea() const
{ return 2*Circle::getArea()+getCircumference() *getHeight();
} // end function getArea

// calculate Cylinder volume
double Cylinder::getVolume() const
{return Circle4::getArea() * getHeight();}// end function getVolume
```

Cylindertest.cpp

```
// output Cylinder object
void Cylinder::print() const
{   Circle::print();   cout << "; Height = " << getHeight();
} // end function CylinderF
```

برنامج الاختبار للصنف Cylinder:

```
// Cylindertest.cpp
// Testing class Cylinder.
#include "stdafx.h"
#include <iostream>
#include <iomanip>
#include "cylinder.h" // Cylinder class definition
using namespace std;
int main()
{   // instantiate Cylinder object
    Cylinder cylinder( 12, 23, 2.5, 5.7 );
```

Cylindertest.cpp

```
// display point coordinates
cout << "X coordinate is " << cylinder.getX()
      << "\nY coordinate is " << cylinder.getY()
      << "\nRadius is " << cylinder.getRadius()
      << "\nHeight is " << cylinder.getHeight();
cylinder.setX( 2 );      // set new x-coordinate
cylinder.setY( 2 );      // set new y-coordinate
cylinder.setRadius( 4.25 ); // set new radius
cylinder.setHeight( 10 ); // set new height
// display new cylinder value
cout << "\nThe new location, radius of circle are\n";
cylinder.print();
// display floating-point values with 2 digits of precision
cout << fixed << setprecision( 2 );
```


Cylindertest.cpp

```
// display cylinder's diameter
cout<<"\n\nDiameter is "<<cylinder.getDiameter();
// display cylinder's circumference
cout<<"\nCircumference is "
    <<cylinder.getCircumference();
// display cylinder's area
cout << "\nArea is " << cylinder.getArea();
// display cylinder's volume
cout << "\nVolume is " << cylinder.getVolume();
system("pause");    return 0; // indicates successful termination
} // end main
```

الخرج:

Multi-level inheritance

4-3- الوراثة متعددة المستويات

الخرج:

X coordinate is 12

Y coordinate is 23

Radius is 2.5

Height is 5.7

The new location, radius of circle are

Center = [2, 2]; Radius = 4.25; Height = 10

Diameter is 8.50

Circumference is 26.70

Area is 380.53

Volume is 567.45 Press any key to continue . . .

Multi-level inheritance

4-3- الوراثة متعددة المستويات

يمكن تلخيص أعضاء الأصناف الثلاثة في مثالنا هذا كما يلي:

	Point		Circle		Cylinder
private	x y		radius		Height
protected					
public	setX() setY() getX() getY() print()		setX() setY() getX() getY() Point::print() setRadius() getRadius() getDiameter() getCircumference() getArea() print()		setX() setY() getX(), getY() Point::print() setRadius() getRadius() getDiameter() getCircumference() Circle::getArea() Circle::print() setHeight() getHeight() getArea() getVolume() print()

انتهت تمارين الأسبوع السادس