

بسم الله الرحمن الرحيم

جدول للمصطلحات البرمجية + مبادئ OOP

Lesson 1 : What is OOP and Why ?

Function يعمل شغلة واحدة فقط

Function Programming (FP) هي مجموعة من Function و Procedure التي تبني البرنامج ،
يتم استدعاءها وقت الحاجة إليها

الفرق بين (FP) Functional vs (OOP) Object Oriented Programming ؟

مثال لإنشاء نظام للجامعة University System

باستخدام **FP** : سيتم كتابة العديد والعديد من **Functions** لأنه مشروع كبير جدا ، فيتم تجزيته الى
Small Function ثم يتم تركيب Function بعضه على بعض مثل **Lego**

عند استخدام (**FP**) في المشاريع الكبيرة : يكون عندك آلاف Functions ستنسى بعض Function
- ك إعادة كتابة Function موجود - + يكونوا غير مرتبين ، وعددهم هائل جدا فلا تستطيع تذكرهم
، طريقة التعامل ب FP في الأنظمة الكبيرة تكون شبه مستحيلة

المشكلة ليست في عدد Functions الهائل وإنما في طريقة تنظيمهم - ونظرتك لهم أو تعاملك معهم
ليست المشكلة في عدد Functions الهائل لطالما أنهم منظمين تستطيع الوصول إليهم بسهولة =

(OOP) Object Oriented Programming

الفرق بين **FP vs OOP** : أن **OOP** تغير نظرتك للكود بشكل أقرب للواقع أو الحياة العملية أي
تجعلك تفكر في تعاملك للكود كأنك تتعامل مع الحياة الواقعية ، تتعامل مع Functions عن
طريق **Object**

باستخدام **OOP** : ماهي **الأشياء أو الكائنات Object** التي تريد برمجتها في الجامعة ؟
(طلاب ، كورس ، موظفين ، دكتور ، الأقسام أو الكليات ، التخصص) كل شيء من هذه الأشياء
يسمى **Object** لذا أنت تبرمج **أشياء Object** وليس **Functions** + أنك تفكر في الكود من فوق الى
تحت أو من الكليات الى الجزئيات وليس العكس
يتم توزيع آلاف أو مئات Functions تحت ما يناسبه - أي له علاقة - في **Object** مثال طلاب
Student Object == يسمى Class Student يندرج تحته كل **Members** - كل شيء له علاقة
ب Student - هي تشبه Structure مبدئيا لسهولة الشرح لكن بإمكانك إضافة **Method** بداخلها

من فوائد استخدام OOP

- أنك تتعامل مع **class** عن طريق **Object** يندرج تحت العديد من Functions
 - تعطيك مفاهيم أو ميزات عديدة تجعلك تختصر كتابة الكود أو إعادة استخدامه
 - تحكم في الوصول – أو عدمه – الى **Method** ل Developer == أمان للكود
 - OOP تعطيك تحكم كامل في الكود وأمان أكثر وإعادة استخدام للكود
- لدى **OOP** المزيد من المفاهيم والطرق التي تتيح لك التعامل مع Code الخاص بك بطريقة أسهل بكثير ، ولديك سيطرة أكبر على Code الخاصة بك

مصطلحات برمجية

Method يكون في **class** وهو **Function & Procedure**

Members هي **Variables & Method**

Paradigms إما **FP** أو **OOP**

#Lesson 2: Classes and Objects

أهمية **OOP** تجعلك ترى الكود من فوق الى تحت أو من الكليات الى الجزئيات وليس العكس ، فأنت ترمج أشياء أو كائن – فأنت تتعامل مع – **Object**

البرمجة الشيئية هي (**OOP**) **Object Oriented Programming**

قبل أن يكون **Object** لابد أن يكون **Class** مثال **Student Object == يسمى Class Student** يندرج تحته كل **Members** – كل شيء له علاقة ب **Student** – هي تشبه **Structure** مبدئيا لسهولة الشرح و بإمكانك إضافة **Method** بداخل **class** وكذلك **struct** – سيدرس الفرق لاحقا

سميت **class** : لأنك صنف الكود الى مجموعات ، مأخوذة من **Classification**

struct stPerson عبارة عن **Datatype** مثله مثل **int , bool , string** عند استدعاء **struct** في **main()** تعرف متغير **Variable** من نوع **stPerson** اسمه **Person**

```
#include <iostream>
using namespace std;

struct stPerson
{
    string FirstName;
    string LastName;
};

int main()
{
    stPerson Person;

    Person.FirstName = "Mohammed";
    Person.LastName = "Abu-Hadhoud";
}
```

class clsPerson عبارة عن **Datatype** مثله مثل مبدئيا لتقريب المعلومة **struct** ،

عند تسمية كل **class** يفضل وضع أول ثلاثة أحرف **cls** ثم وضع اسم خاص ب **class** لسهولة الرجوع الى **classes** بسرعة

عند استدعاء **class** في **main()** أو غيره تعرف متغير **Variable** من نوع **clsPerson** اسمه مثلا **Person1** يسمى **Object** أو **Instance**

كل شيء داخل **class** يكون بشكل افتراضي – By Default – مخصص Private لهذا **class** فقط يعني لا تستطيع استدعاؤه من خارج **class**

لجعل **Members** يمكن الوصول إليها من خارج **class** تجعله عام: **public** - أي شيء تحت هذا **public**: تستطيع الوصول إليه من أي مكان في البرنامج

الفرق ما بين **class** و **Object** ؟

Class هي المعرف الذي بداخله الكود ويعتبر هو Datatype لذلك عند استخدامه لابد من تعريف Variable ويسمى **Object** – شيء واحد –

String أكبر مثال على **class** ، كلما عرفت Variable من نوع **String** تكون عرفت **Object** من نوع **String**

```
#include <iostream>
using namespace std;
// Lesson #2

class clsPerson
{
    // Class هو صنف تجعل كل شيء له علاقة بهذا الصنف تحت هذا Class

    //this private and will not be accessed from outside the class
    //for internal use only

    // كل شيء داخل Class يكون بشكل افتراضي By Default هو خاص Private في هذا Class
    // يعني لا تستطيع الوصول إلى Members إلا من داخل Class لا تستطيع ذلك من main() أو غيره
    private: // private هو لا سواء كتيبه أم لا فهو
    int x;

    // جعل Members يمكن الوصول إليها من خارج Class يتم تعريف
    public:
        string FirstName;
        string LastName;

        // Class يمكن إضافة Method في
        string FullName()
        {
            return FirstName + " " + LastName;
        }
};

int main()
{
    clsPerson Person1;

    Person1.FirstName = "Mohammed";
    Person1.LastName = "Abu-Hadhoud";

    cout << Person1.FullName() << endl;

    // S1 هو Object من Class اسمه
    string S1;
    S1 = "Saeed";
    // S1. = string تظهر Method الموجودة داخل Class الذي اسمه string
    cout << S1 << "\n Size = " << S1.size();
}
```

Quiz Answers		Question
Class is the blue-print of Object	Class is a Datatype	What is Class ?
No , You have to declare Object of the Class first , and access all members and Methods through the Object not Class		Can you access Class members and methods directly ?
ObjectName.FunctionName();		How to access member function of class using Object ?
No , Only Public Members and Methods can be accessed through the Object , all Private Members and Methods are for internal use inside the Class		If you have Private Member or Method in Class , Can you access this Member or Method through Object ?
True		Any Function or Procedure inside Class is called "Method"
True		Object is Instance of Class
1. C is procedural / Functional programming language 2. C does not support OOP , While C++ Support OOP 3. C++ Supports Procedural / Functional Programming And OOP as well		What is the difference between C & C++ ?
True		Class Members means any Variable or Function inside the Class is called "Member"

#Lesson 3: Class Members

Class هو فئة تندرج تحته كل **Members == Function & Procedure & Variable** التي لها علاقة بهذه الفئة ، وهذا **Members** التي بداخل **Class** إما يكون **Public** : تستطيع مناداته أو استخدامه من أي مكان في البرنامج أو يكون **Private** لا تستطيع مناداته إلا من داخل **Class**

Class عبارة عن Datatype ، من أجل استخدام هذا **Class** الذي هو عبارة عن Template & Form فارغ ، لابد من تعريف Variable من نوع **Class** عن طريق **Object** و **Object** لا يأتي إلا من **Class** (مثل تعريف أي Variable من نوع Datatype)

Object فيه محتويات وتسمى **Members**

```
clsPerson Person1;  
Person1.FirstName
```

Members ينقسم الى قسمين

- ❖ **Data Members** يعني أي Variable تستطيع تخزين البيانات فيه داخل **Class**
- ❖ أي Function & Procedure يسمى **Member (Method or Function)**

```
class clsPerson  
{  
private: // private كتيبه أم لا فهو  
int x;  
  
public:  
    // Data Members هي Global Variable في داخل Class وتسمى  
    string FirstName;  
    string LastName;  
  
    // Member ( Method or Function ) ويسمى  
    string FullName()  
    {  
        return FirstName + " " + LastName;  
    }  
};
```

Quiz	Answers	Question
True		Data Member is any Variable inside the Class that holds data
True		Function Member is any Function or Procedure inside a Class
True		Class Members are Data Members and Function Member

#Lesson 4: Object In Memory

عند تعريف **Object** من نوع **Class** تستطيع تعريف أكثر من **Object**

Class مكونة من **Members** وينقسم الى **Data Members** و **Function Members**

كل **Object** يتم تعريفه يكون له مكان خاص في Memory لتخزين **Data Members**
أما **Function Members** يكون لهم مكان واحد في Memory مهما تعدد **Objects**

Object يفصل Data في Memory بين كل **Object** مثل **Person1**, **Person2**

```
#include <iostream>
using namespace std;
class clsPerson
{
    // public: يمكن الوصول إليها من خارج Class يتم تعريف
    public:
        // هذه المتغيرات هي Global Variable في داخل Class وتسمى Data Members
        string FirstName;
        string LastName;

        // يمكن إضافة Method في Class ويسمى ( Method or Function )
        string FullName()
        {
            return FirstName + " " + LastName;
        }
};

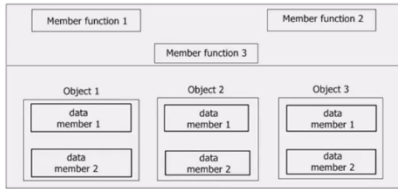
int main()
{
    clsPerson Person1, Person2;

    Person1.FirstName = "Mohammed";
    Person1.LastName = "Abu-Hadhoud";

    Person2.FirstName = "Saeed";
    Person2.LastName = "Omar";

    cout << "Person1 : " << Person1.FullName() << endl;
    cout << "Person2 : " << Person2.FullName() << endl;
}
```

Each instance has its own space in memory, Only Member functions are shared among all objects



```
#include <iostream>
using namespace std;

class clsPerson
{
public:
    string FirstName;
    string LastName;

    string FullName()
    {
        return FirstName + " " + LastName;
    }
};

int main()
{
    clsPerson Person1, Person2;

    Person1.FirstName = "Mohammed";
    Person1.LastName = "Abu-Hadhoud";
    Person2.FirstName = "Ali";
    Person2.LastName = "Maher";

    cout << "Person1: " << Person1.FullName() << endl;
    cout << "Person2: " << Person2.FullName() << endl;
}
```



Quiz	Answers	Question
false		Every Object has it's own space in Memory that hold both (Data / Function) Members
True		Every Object has it's own space in Memory that holds only Data Members
True		Function Members are shared to all Object in Memory and has one space for them

OOP تنقل تفكيرنا الى مستوى آخر في كتابة الكود ، وممن مميزاتنا تعطيك تحكم Control على الكود - تستطيع جعل شروط على الكود ومنها : Access Specifiers (Modifiers)

Class مكونة من **Members** وينقسم الى **Data Members** و **Function Members**

تستطيع التحكم في **Members** من يستطيع مناداتهم ومن لا يستطيع + عدم التعديل عليه
Access Specifiers (Modifiers) يعطيك أمان للكود

من الذي يستفيد - يستطيع الوصول الى - من **Members**

- ❖ داخل **Class** نفسه
- ❖ أي أحد من خارج **Class** عن طريق **Object**
- ❖ كل **Classes** التي ترث **Inheritance** هذه **Class** - سدرس لاحقا -

يوجد ثلاثة أنواع من **Access Specifiers (Modifiers)** ل **Members** بمعنى آخر من الذي له الصلاحية للوصول الى **Members**

١. **Private** : تستطيع الوصول الى **Members** من داخل **Class** فقط
 ٢. **Protected** : يكون داخل **Class** فقط + كل **Classes** التي ترث **Inheritance** هذه **Class**
 ٣. **Public** : أي أحد يستطيع الوصول إلى **Members** من أي مكان
- Private** هو للاستخدام الداخلي ل **Class** فقط أو لحماية **Members** من التعديل عليه من الخارج

```

#include <iostream>
using namespace std;

class clsPerson
{
private: // سواء كتبتها أو لا فهي private يفضل كتابتها
    //only accessible inside this class
    // هؤلاء Members للاستخدام الداخلي فقط في هذا Class
    // عادة يكون عندك الكثير من Members تستخدمها لتجهيز Class
    // عدم السماح لأي Developer من التعديل على هذا private + أمان للكود
    int Variabl1=5;

    int Function1()
    {
        return 40;
    }

protected: //only accessible inside this class and all classes inherits this
class
    int Variabl2=100;

    int Function2()
    {
        return 50;
    }

public: //accessible for everyone outside/inside/and classes inherits this class

    string FirstName;
    string LastName;

    string FullName()
    {
        return FirstName + " " + LastName;
    }

    float Function3()
    {
        // Class من داخل protected أو private سواء Members تستطيع مناداة
        return Function1() * Variabl1 * Variabl2;
    }

};

int main()
{
    // Lesson #5

    clsPerson Person1;

    Person1.FirstName = "Mohammed";
    Person1.LastName = "Abu-Hadhoud";

    cout << "Person1: " << Person1.FullName() << endl;

    cout << Person1.Function3();

}

```

Quiz Answers	Question
False	Access modifiers (or access specifiers) are keywords in object-oriented languages that set the accessibility of .classes, methods, and other members
Private .1 Protected .2 Public .3	Which of the following is Access Specifiers / Modifiers:
True	Public Members can be accessed from inside and outside the class
False	Private Members can be accessed from outside the class through object
False	Private Members can be accessed by any class inherits the current class
True	Private Members can be accessed only from inside the class, it cannot be accessed from outside the class nor from the classes inherits the current class
Protected	If you want to have a member that is private to outside class and public to classes inherits the current class, which access specifier/modifier you use
False	Protected Members can be accessed from outside class through objects
True	Protected Members can be accessed from inside class and from all classes inherits the current class
True	OOP is more secured because you can hide members from developers
True	Inside the class I can access everything including Public, Private , and Protected Members

#Lesson 6 : Properties Set and Get

في الدرس الماضي كان يتم التعديل على **Public Variable** مباشرة من **Object** مثال :
`Person1.FirstName = "Mohammed";` على **Class** ، وليس عن طريق **Function** في **Class**

Property من أهم الأشياء الموجودة في **OOP**

قاعدة : لا تعدل أو تغير أو تستدعي أي **Public Variable إلا عن طريق **Function = Property** ، كل **Public Variable** في **Class** يتم إنشاء **Two Functions** له**

١. **للتعديل على Variable** ويسمى **Property Set**

٢. **للحصول على Variable** ويسمى **Property Get**

بمعنى آخر : لا يفضل تعريف **Public Variable في **Class****

يفضل وضع شرطة سفلية في بداية اسماء **Members** للدلالة على أنها من نوع **Private** لتصل إليها بشكل أسرع

يفضل في **OOP** عدم كتابة كود خارج **Class** لأنه قد تحتاج هذا الكود في نظام آخر مثلاً

من فوائد **Property**

❖ تعطي أمان ل **Data Members**

❖ لا يسمح للتعديل إلا بإذن

❖ يزيد من إعادة استخدام الكود

مصطلح **Audit Trailing** هو :

حفظ القيم القديمة ثم التعديل عليها سواء في ملف أو **Database**

Property هي عبارة عن **Two Functions : Set & Get** ويتم تخزين **Private Members** في **Property** ولا يتم الوصول إلى **Private** إلا من داخل **Class** فقط ،
Private هو **Global** بالنسبة ل **Class** ليتم الوصول إليه عن طريق **Members**

```
#include <iostream>
using namespace std;
```

```

class clsPerson
{
private: // سواء كتبتها أو لا فهي private يفضل كتابتها
        // Class Members خارج هذه Class
        // Class خارج Class
        string _FirstName;
        string _LastName;

public:
        // Function على Variable في public باستخدام
        // Property Set ويسمى
        void SetFirstName(string FirstName)
        {
            // جعلت قيود للتعديل على Variable عن طريق Property
            _FirstName = FirstName;
        }

        // Function الحصول على Variable في public باستخدام
        // Property Get ويسمى
        string FirstName()
        {
            return _FirstName;
        }

        //Property Set
        void SetLastName(string LastName)
        {
            _LastName = LastName;
        }

        //Property Get
        string LastName()
        {
            return _LastName;
        }

        string FullName()
        {
            return _FirstName + " " + _LastName;
        }
};

int main()
{
    clsPerson Person1;

    // Person1 = Object يتم تخزينها في
    // Function = SetFirstName عن طريق Class في Variable = _FirstName يتم الوصول الى
    Person1.SetFirstName("Mohammed");
    Person1.SetLastName("Abu-Hadhoud");

    cout << "First Name : " << Person1.FirstName() << endl;
    cout << "Last Name : " << Person1.LastName() << endl;
    cout << "Full Name : " << Person1.FullName() << endl;

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	Properties are Functions allow you to Update Private Members inside the class	
True	Properties are two functions one for Setting Data and One for getting Data	
True	If you want to update data inside class you should write a property function to set them	
True	If you want to retrieve Data Member from a class you should write a property function to get that data	
Ture	Both property functions set and get they use a private data member to store and get data from it	

سؤال : كيف تستطيع عمل Read Only Property ؟

مثال : السماح ل Developer للوصول الى ID Person للقراءة فقط ، ولا يسمح له بالتعديل عليها

#Lesson 7 : Read Only Property

Property تنقسم الى قسمين

Property Set ❖

Property Get ❖

وهما يخزنا **Private Members** بنفس اسم **Members** تقريبا في **Class** ، و **Private** لا يتم الوصول إليه من خارج **Class** لذا فهو **Security Data**

إذا تريد التعديل على **Members** فتضعها في **Property** نوع **Set** فيه باراميتري يخزن في **Private _Members**

عندما تستخدم نوع واحد من **Property** مثل **Get** يكون للقراءة فقط **Read Only Property**

عندما تستخدم نوع واحد من **Property** مثل **Set** يكون للكتابة فقط **Write Only Property**

```
#include <iostream>
using namespace std;

class clsPerson
{
private: // سواء كتبتها أو لا فهي private يفضل كتابتها
    int _ID = 10; // تستطيع جلب البيانات من قواعد البيانات وتخزينها في متغير
public:

    //Property Get, this is a read only property because we don't have a s
    int ID()
    {
        return _ID;
    }
}
```

Answers	Question	Quiz
False	In order have a read only property you only implement the set function and you don't implement the get function	
Get Function Only	In order have a read only property you implement	
Set Function Only	In order to have a write only property you implement	
Both Set and Get Functions	In order have a read/write property you implement	

#Lesson 8 : Properties Set and Get through "="

تستطيع الوصول الى **Private Members** من خارج **Class** المتواجدة في **Public Property** عن طريق **Object** – بحيث تنادي **Two Functions** إما **Set** أو **Get** –

تختصر **Two Functions** الى **One Function** فقط بحيث اذا استدعي **Function** يعني **Get** ، أما اذا استدعي **Function** + = يعني **Set**

كل Property Set & Get في البرنامج لابد له من إضافة هذا : الكود الذي بالأسفل

`_declspec(property(get = GetName, put = SetName)) string Name;`
هذا الكود هو **Class** جاهز في **C++** ، `string Name` لابد ان يكون **Datatype** مثل `GetName` `string Name` هو الاسم الذي ستعامل به في **Object**. ويظهر بجانبه إشارة مفتاح وتسمى **Property** - ويسمى هذا السطر **Declaration Specification**

```
#include <iostream>
using namespace std;

class clsPerson
{
private: // سواء كتبها أو لا فهي private يفضل كتابتها
    string _FirstName;

public:
    // تستطيع التعديل على Variable في public باستخدام Function
    // Property Set ويسمى
    void SetFirstName(string FirstName)
    {
        // جعلت قيود للتعديل على Variable عن طريق Property
        _FirstName = FirstName;
    }
    // Function الحصول على Variable في public باستخدام Function
    // Property Get ويسمى
    string GetFirstName()
    {
        return _FirstName;
    }

    // دمج في Set & Get Functions
    _declspec(property(get = GetFirstName, put = SetFirstName)) string
    FirstName;
};

int main()
{
    clsPerson Person1;

    // instead of the above we only write this
    Person1.FirstName = "Mohammed";
    cout << Person1.FirstName;

    system("pause>0");
    return 0;
}
```


OOP لها أكثر من مبدأ أو مفهوم منها : Encapsulation

الذي تعلمناه في الدروس السابقة أن : **Class** هو فئة يندرج تحته كل **Function & Procedure or Variable** التي لها علاقة بهذه الفئة وتسمى **Encapsulation**

Encapsulation : مأخوذة من كبسولة الدواء البلاستيكية التي هي **Class** التي تجمع كل **Methods** ذات العلاقة تحت سقف واحد ولها جزئين :

١. **Methods** وتسمى **Member Method (Function)**

٢. **Variables** وتسمى **Data Members**

ولا تستطيع الوصول إليهما من خارج **Class** إلا عن طريق **Object**

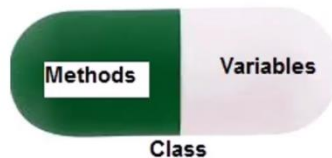
Encapsulation له علاقة في إخفاء البيانات التي هي **Property**

Answers	Question	Quiz
True	In normal terms Encapsulation is defined as wrapping up of data and information under a single unit. In Object Oriented Programming, Encapsulation is defined as binding together the data and the functions that manipulates them	

Encapsulation in Object Oriented (OOP)



```
StudentObject:
Student1.Name
Student1.Email
Student1.GetEnrolledCourses()
Student1.CalculateAverage()
Student1.SendEmail(Subject, Body)
Student1.EnrollInCourse(10)
Student1.PayFees
Student1.UnEnrolFromCourse(10)
```



You can do whatever you want with a certain student only through its object



#Lesson 10 : Second Principle / Concept of OOP : Abstraction

Abstraction هذا المفهوم مهم جدا جدا في البرمجة – ستتم دراسته نظريا –

مثال لتصوير صورة في هاتفك : تفتح التطبيق ثم تضغط على زر التصوير
هذا الزر عبارة عن Function اسمه Take Picture ، وفي هذا Object يوجد Functions كثيرة
أخرى تم إخفاؤها عنك إما بسبب Security وليس هو السبب الرئيسي وإنما أخفوها لأنك
كمستخدم لا تهتمك مثل الأشياء المعقدة التي لا تفيدك

مثال آخر String S1.size() تظهر Method بعد . التي تحتاجها في String
يوجد في Class String تقريبا ضعف Method الظاهرة في object التي تستخدم داخل Class
لكنها لا تفيدك أو لا تهتمك كمستخدم

من أوائل من سيستخدم Class الخاص بك عن طريق Object هم المبرمجين لذلك تظهر لهم
Method التي يحتاجونها فقط

Abstraction تظهر لك فقط Art tribute وهي Method & Property عناصر Class التي
تهتمك – المطور أو المستخدم – فقط

في OOP تفكر أيضا في تصميم البرنامج ومن سيستخدم هذه Class الخاصة بك ، وهدف المطور هو
تسهيل حياته أو حياة الآخرين

Abstraction يختلف تماما عن Abstract Class – ستدرس لاحقا –

Answers	Question	Quiz
True	In simple terms, abstraction “displays” only the relevant attributes of objects and “hides” the unnecessary details	
Through Private Members Only	You Achieve Abstraction ?	

#Lesson 12 : Project 1 : Calculator (Solution)

#Lesson 11 : Project 1 : Calculator (Requirements)

Result After Adding 10 is: 10

Result After Adding 100 is: 110

Result After Subtracting 20 is: 90

Result After Dividing 0 is: 90

Result After Dividing 2 is: 45

Result After Multiplying 3 is: 135

Result After Cancelling Last Operation 0 is: 45

Result After Clear 0 is: 0

```
#include <iostream>
using namespace std;

// هذه Class يندرج تحتها كل Members التي تتعلق ب Calculator
class clsCalculator
{
private :
    // تحت هذا private لا يستطيع مناداتهم إلا من داخل هذه Class وتسمى Abstraction
    float _Result = 0;
    float _LastNumber = 0;
    string _LastOperation = "Clear";

    // يخزن النتيجة السابقة لآخر عملية
    float _PreviousResult = 0;
    bool _IsZero(float Number)
    {
        return (Number == 0);
    }

public:
    // هذه كل Method التي يستطيع المستخدم استخدامها في Object
    void Add(float Number)
    {
        _LastNumber = Number;
        _PreviousResult = _Result;
        _LastOperation = "Adding";
        _Result += Number;
    }
}
```

```

void Subtract(float Number)
{
    _LastNumber = Number;
    _PreviousResult = _Result;
    _LastOperation = "Subtracting";
    _Result -= Number;
}

void Divide(float Number)
{
    _LastNumber = Number;

    if (_IsZero(Number))
    {
        Number = 1;
    }

    _PreviousResult = _Result;
    _LastOperation = "Dividing";
    _Result /= Number;
}

void Multiply(float Number)
{
    _LastNumber = Number;
    _LastOperation = "Multiplying";
    _PreviousResult = _Result;
    _Result *= Number;
}

float GetFinalResults()
{
    return _Result;
}

void Clear()
{
    _LastNumber = 0;
    _PreviousResult = 0;
    _LastOperation = "Clear";
    _Result = 0;
}

void CancelLastOperation()
{
    _LastNumber = 0;
    _LastOperation = "Cancelling Last Operation";
    _Result = _PreviousResult;
}

void PrintResult()
{
    cout << "Result ";
    cout << "After " << _LastOperation << " " << _LastNumber << " is: "
<< _Result << "\n";
}
};

```

```

int main()
{
    // Class عبارة عن نسخة من Object / Instance من نوع Calculator1
    // لا تستطيع الوصول الى Class إلا عن طريق Object

    clsCalculator Calculator1;

    // هذه Object توصلك الى Method الموجودة في Class من نوع Public
    // هذه Method التي تظهر للمستخدم هي التي يحتاجها أو التي تهتم فقط وهذه تسمى Abstraction

    Calculator1.Clear();

    Calculator1.Add(10);
    Calculator1.PrintResult();

    Calculator1.Add(100);
    Calculator1.PrintResult();

    Calculator1.Subtract(20);
    Calculator1.PrintResult();

    Calculator1.Divide(0);
    Calculator1.PrintResult();

    Calculator1.Divide(2);
    Calculator1.PrintResult();

    Calculator1.Multiply(3);
    Calculator1.PrintResult();

    Calculator1.CancelLastOperation();
    Calculator1.PrintResult();

    Calculator1.Clear();
    Calculator1.PrintResult();

    system("pause>0");
    return 0;
}

```

#Lesson 13 : Constructors

Constructors هو موجود في أي **Class** يكون معرف في داخل **Class** بشكل تلقائي default Constructor ، إلا إذا تم تعريفه بشكل يدوي Override ألغى default Constructor أكبر غلط في **Object** أنه يتم استدعاؤه ولا يتم أخذ البيانات أولاً ، أقل شيء يتم تعريف قيمة مبدأيه **Constructors** أي – العامل – الذي يبني مأخوذ من Construction أي البناء

Constructor هو Function يتم استدعاؤه كلما عرفت **Object** من **Class** ، يتم إنشاؤه بشكل افتراضي default **Constructor** عن طريق Compiler – عادة يكون فارغ – لا يوجد فيه كود أو تعرفه أنت في **Class** ويكون اسمه مثل اسم **Class** الذي أنشأته

```
clsAddress()
{
    // هذا مثال للتوضيح من أنه يتم استدعاؤه مع كل Object
    cout << "Hi I'm the Constructor. ";
}
```

– **Constructor** ينادى مع كل **Object** يتم إنشاؤه - كيف تستطيع الاستفادة منه ومنها

- ❖ وضع قيم مبدأيه أو Overloading أو إرجاع قيمة
- ❖ جلب البيانات من Database أو من File ... وتحميلهم في **Object** أو مناداته مع بارامتر

```
#include <iostream>
using namespace std;

class clsAddress
{
private:
    string _AddressLine1;
    string _AddressLine2;
    string _POBox;
    string _ZipCode;

public:
    // Constructor يسمى
    // إذا لم تكتبه ف Compiler يكتبه عنك
    // مثال clsAddress() { }
    clsAddress(string AddressLine1 , string AddressLine2 , string POBox , string
ZipCode)
    {
        // عند استدعاء هذه Class في Object لابد من تعبئة هذه البارامتر
        // وهذه البارامتر المعينة يتم تخزينها في private Variable
        // أجريت أي أحد Class أن يدخل قيم قبل أن يستعمل Method في Object
        // يستخدم هذه
        _AddressLine1 = AddressLine1;
        _AddressLine2 = AddressLine2;
        _POBox = POBox;
        _ZipCode = ZipCode;
    }
}
```

```

void SetAddressLine1(string AddressLine1)
{
    _AddressLine1 = AddressLine1;
}

string AddressLine1()
{
    return _AddressLine1;
}

void SetAddressLine2(string AddressLine2)
{
    _AddressLine2 = AddressLine2;
}

string AddressLine2()
{
    return _AddressLine2;
}

void SetPOBox(string POBox)
{
    _POBox = POBox;
}

string POBox()
{
    return _POBox;
}

void SetZipCode(string ZipCode)
{
    _ZipCode = ZipCode;
}

string ZipCode()
{
    return _ZipCode;
}

void Print()
{
    cout << "\nAddress Details:\n";
    cout << "-----";
    cout << "\nAddressLine1: " << _AddressLine1 << endl;
    cout << "AddressLine2: " << _AddressLine2 << endl;
    cout << "POBox          : " << _POBox << endl;
    cout << "ZipCode         : " << _ZipCode << endl;
}

};

int main()
{
    clsAddress Address1("Queen Alia Street", "B 303 ", "11192", "5555");

    Address1.Print();

    // لا تستطيع استخدام Object عنج عدم وضع قيم للباراميتر في Object
    // رسالة الخطأ هي 'clsAddress' لا يوجد معالج افتراضي
    // clsAddress Address2;

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	A constructor is a special type of member function that is called automatically when an object is created	
True	In C++, a constructor has the same name as that of the class and it does not have a return type	
<ul style="list-style-type: none"> • Have the same name of the class • Should not return type • Should be Public 	Constructor should	
True	Default Constructor: A constructor with no parameters is known as a default constructor	
No , because if you don't write it the Compiler will write it for you	You should always write Default Constructor	
True	Parameterized Constructor: a constructor with parameters is known as a parameterized constructor. This is the preferred method to initialize member data	
True	When you have a parameterized constructor it will override the default constructor	
True	When you have a parameterized constructor it will override the default constructor	

#Lesson 14 : Copy Constructors

Copy Constructor يتم إنشاؤه default عن طريق Compiler لا داعي لكتابته لأن Compiler يكتبه عنك بشكل افتراضي

```
#include <iostream>
using namespace std;
class clsAddress
{
private:
    string _AddressLine1;
    string _AddressLine2;
    string _POBox;
    string _ZipCode;
public:
    // Constructor يسمى
    // إذا لم تكتبه ف Compiler يكتبه عنك
    // مثال { } clsAddress()
    clsAddress(string AddressLine1 , string AddressLine2 ,string POBox ,string
ZipCode)
    {
        // عند استدعاء هذه Class في Object لابد من تعبئة هذه البارامتر
        // وهذه البارامتر المعينة يتم تخزينها في private Variable
        // أجريت أي أحد يستخدم هذه Class أن يدخل قيم قبل أن يستعمل Method في Object
        _AddressLine1 = AddressLine1;
        _AddressLine2 = AddressLine2;
        _POBox = POBox;
        _ZipCode = ZipCode;
    }
    // Copy Constructor
    // يتم إنشاؤه default عن طريق Compiler لا داعي لكتابته
    // يكون ل Constructor نفس اسم Class لكن يكون الاختلاف في البارامتر
    clsAddress(clsAddress & old_obj)
    {
        // هذه طريقة كتابة Copy Constructor
        _AddressLine1 = old_obj.AddressLine1();
        _AddressLine2 = old_obj.AddressLine2();
        _POBox = old_obj.POBox();
        _ZipCode = old_obj.ZipCode();
    }
    // نفس كود الدرس السابق
}
int main()
{
    clsAddress Address1("Queen Alia Street", "B 303 ", "11192", "5555");
    Address1.Print();

    // تعريف Object2 من نفس نوع Object1 التي فيها البيانات
    // ثم نسخ كل بيانات Object1 في Object2
    clsAddress Address2 = Address1;

    // نتيجة Object2 تماما مثل Object1
    Address2.Print();

    system("pause>0");
    return 0;
}
```

Answers	Question	Quiz
True	The copy constructor is used to initialize the members of a newly created object by copying the members of an already existing object	
True	The process of initializing members of an object through a copy constructor is known as copy initialization	
True	It is also called member-wise initialization because the copy constructor initializes one object with the existing object, both belonging to the same class on a member-by-member copy basis	
True	The copy constructor can be defined explicitly by the programmer. If the programmer does not define the copy constructor, the compiler does it for us	
No , because the compiler will do it for you	You should always implement a copy constructor in your code	
Default Constructor • Parameterized • Constructor Copy Constructor •	What are the types of constructors	
Yes , You can using function overloading and this is called "Constructor Overloading"	Can you have more than one constructor in a class	

#Lesson 15 : Destructors

Constructor هو عكس **Destructor** وهو المدمر ل **Members** وكذلك **Object** بعد الانتهاء منه ، فيتم تدميره – إتلافه عند خروجه من النطاق – من الذاكرة وهو **Destructor** آخر **Method** تتم مناداته في **Object** قبل تدميره ، ويتم إنشاؤه مرة واحدة فقط في **Class** :
طريقة كتابة **Destructor** هي **~ClassName**

يتم تنفيذ التدمير **Destructor** بعد الانتهاء أو الخروج من كامل Function – سواء **main()** أو غيره

من فوائد **Destructor**

- ❖ حفظ البيانات – معينة – في Database أو في File
- ❖ غلق Connection
- ❖ Releasing ل Object آخر – قد يكون عندك **Class** بداخله **Class Object** آخر تم تعريفها **Pointer** يتم حذفها في **Destructor** داخل **Class**

قاعدة في C++ : عند استخدام **New Pointer** استخدام معها **delete**

```
#include <iostream>
using namespace std;

class clsPerson
{
public:
    string FullName;

    //This is Instructor will be called when object is built.
    clsPerson()
    {
        // ينادى في البداية عند إنشاء Object
        FullName = "Mohammed Abu-Hadhoud";
        cout << "\nHi, I'm Constructor";
    }

    //This is destructor will be called when object is destroyed.
    ~clsPerson()
    {
        // ينادى عند الانتهاء من Object
        cout << "\nHi, I'm Destructor";
    }
};

void Fun1()
{
    clsPerson Person1;
    //after exiting from function, person1 will be
    //destroyed and destructor will be called.
}
```

```

void Fun2()
{
    // delete عند استخدام Pointer لا يتم حذفها من الذاكرة إلا بشكل يدوي باستخدام
    clsPerson* Person2 = new clsPerson;
    //always use delete whenever you use new, otherwise object will remain in
memory
    // عند استخدام delete حينها ينادى Destructor لتدميره
    delete Person2;
}

int main()
{
    Fun1();
    Fun2();

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	Destructor is an instance member function which is invoked automatically whenever an object is going to be destroyed. Meaning, a destructor is the last function that is going to be called before an object is destroyed	
True	Destructor has the same name as their class name preceded by a tilde (~) symbol	
True	It is not possible to define more than one destructor	
True	The destructor is only one way to destroy the object create by constructor. Hence destructor can-not be overloaded	
True	Destructor neither requires any argument nor returns any value	
True	It is automatically called when object goes out of scope	

#Lesson 16 : Static Members (Variable)

Static حياته حياة البرنامج – لا يتم تدميره عند الخروج من Function إذا كان بداخله –

مثال عندك **Object** من نوع **Class** يوجد بداخله **Variable** عند تعريف 10 **Objects** لكل **Object** له مكان خاص في الذاكرة مخزن في كل واحد منهم **Variable** خاص به -لا علاقة له ب **Object** آخر

إذا عرفت **Variable** لا بد أن يكون من نوع **Static** يخزن له مكان واحد في الذاكرة لكل 10 **Objects** - يكون **Variable** (Static / Share) مشترك بين **Objects** 10-

عندما يكون **Static Variable** من نوع **Public** تستطيع التعديل عليه من أي **Object** فياثر هذا التعديل على كل **Objects** (تكون قيمة واحدة **Variable** لجميع **Objects**) لأن التعديل تم على مكان واحد في الذاكرة

بعد تعريف `static int counter;` داخل **Class** لابد من تعريفه تحت أو خارج **Class** لا يتم تعريفه داخل أي **Function** بل يعرف **Global Variable**
`int clsA::counter = 0; //static variable initialization outside the class`

في C++ يسمى **Static Variable** وفي غيرها – C# - يسمى **Shared Variable**

Static هو متغير مشترك لجميع **Objects** ، وأي **Object** يعدله يتم تعديله لجميع **Objects** الأخرى

```
#include <iostream>
using namespace std;

class clsA
{
public:
    // لو عندك 10 Object يكون عندك 10 var في الذاكرة
    int var;

    // لو عندك 10 Object يكون عندك counter واحد فقط في الذاكرة لكل 10 Object
    // يكون counter تابع ل Class وليس ل Object
    // تكون حياته حياة كامل البرنامج
    static int counter;

    // Constructor يسمى
    clsA()
    {
        // كلما يتم تعريف Object جديد يزيد counter واحد
        counter++;
    }

    void Print()
    {
        cout << "\nvar    = " << var << endl;
        cout << "counter = " << counter << endl;
    }
};
```

```
// Class تعريف static int counter داخل Class لابد من تعريفه تحت أو خارج Class
// Global Variable بل يعرف Function داخل أي Function
int clsA::counter = 0; //static variable initialization outside the class
```

```
int main()
{
    // Lesson #16

    // Counter 3 == Object 3 تم تعريف
    clsA A1, A2, A3;

    A1.var = 10;
    A2.var = 20;
    A3.var = 30;

    A1.Print();
    A2.Print();
    A3.Print();

    // Object التعديل على static counter من أي Object
    // ف يتم التعديل على كل Objects لأن التعديل تم على مكان واحد في الذاكرة
    A1.counter = 500;

    cout << "\nafter changing the static member counter in one object:\n";

    A1.Print();
    A2.Print();
    A3.Print();

    system("pause>0");
    return 0;
}
```

Answers	Question	Quiz
True	Static Member is a variable that is shared for all objects, any object modifies it get modified for all other objects	
False	Each Object has it's own static members	
True	Static members are on the class level not for each object	
True	Static Members are accessible from all objects	

#Lesson 17 : Static Methods (Functions)

تعلمنا في درس **Encapsulation** : أنك لا تستطيع مناداة **Members** إلا عن طريق **Object** ،
تستطيع مناداة **Static** (Function & Variable) من غير تعريف **Object** – عن طريق **Class** –

```
// عن طريق Class
clsA::Variable;
clsB::Function1();
```

```
// عن طريق Object
clsA A1;
A1.counter ;
clsB B1;
B1.Function1();
```

```
#include <iostream>
using namespace std;
class clsB
{
public:
    // تستطيع مناداة هذا Function من غير تعريف Object
    static int Function1()
    {
        return 10;
    }

    int Function2()
    {
        return 20;
    }
};

int main()
{
    //The following line calls static function directly via class not through
    the object
    cout << clsB::Function1() << endl;

    // عن طريق Object
    clsB B1, B2;

    //At class level you can call only static methods and static members
    // static methods can also be called through the object

    cout << B1.Function1() << endl;
    cout << B1.Function2() << endl;
    cout << B2.Function1() << endl;

    system("pause>0");
    return 0;
}
```

Answers	Question	Quiz
True	Static Functions can be called at class level without a need to have an object	
False	Static Functions cannot be called through object	
True	Static Functions can be called through any object as well as through the class itself	
No. Static methods can only access static members, because static methods can be called at class level without objects, and non static members you cannot access them without having object first	If you have a static function, can you access a non-static members of the class from inside that function	

#Lesson 19 : Person Exercise (Solution)

الحلول في OOP تكون متشابهة جدا لأنها تقرب طرق التفكير

#Lesson 18 : Project 2 : Person Exercise (Requirements)

:Info

ID : 10

FirstName: Saeed

LastName : Omar

Full Name: Saeed Omar

Email : My@gmail.com

Phone : 057842937

The following message sent successfully to email : My@gmail.com

Subject : Hi

? Body : How are you

The following SMS sent successfully to phone : 057842937

? How are you

```
#include <iostream>
using namespace std;
```

// String مثل Members تكون مخزنة في مكتبة تستطيع فقط الوصول الى
// Web | Desktop أي تطبيق آخر مثل

```

class clsPerson
{
    // private حتى لو لم تكتب فهي private
private:
    int _ID;
    string _FirstName;
    string _LastName;
    string _Email;
    string _Phone;

public:
    // ( Parameterize Constructor ) ما بين الأقواس يسمى
    clsPerson(int ID, string FirstName, string LastName , string Email ,string
Phone)
    {
        // Constructor هو أول Function يتم استدعاؤه عند إنشاء Object
        // Initialization تعبئة Object بالبيانات قبل استخدام Members
        _ID = ID;
        _FirstName = FirstName;
        _LastName = LastName;
        _Email = Email;
        _Phone = Phone;
    }
    // Read Only Property
    int ID()
    {
        return _ID;
    }

    // Property Set
    void setFirstName(string FirstName)
    {
        _FirstName = FirstName;
    }
    // Property Get
    string FirstName()
    {
        return _FirstName;
    }

    // Property Set
    void setLastName(string LastName)
    {
        _LastName = LastName;
    }
    // Property Get
    string LastName()
    {
        return _LastName;
    }

    string FullName()
    {
        return _FirstName + " " + _LastName;
        ;
    }

    // Property Set
    void setEmail(string Email)
    {
        _Email = Email;
    }
}

```

```

// Property Get
string Email()
{
    return _Email;
}

// Property Set
void SetPhone(string Phone)
{
    _Phone = Phone;
}

// Property Get
string Phone()
{
    return _Phone;
}

void Print()
{
    cout << "\nInfo:";
    cout << "\n-----";
    cout << "\nID      : " << _ID;
    cout << "\nFirstName: " << _FirstName;
    cout << "\nLastName : " << _LastName;
    cout << "\nFull Name: " << FullName();
    cout << "\nEmail    : " << _Email;
    cout << "\nPhone    : " << _Phone;
    cout << "\n-----\n";
}

void SendEmail(string Subject, string Body)
{
    cout << "\nThe following message sent successfully to email : " <<
_Email ;
    cout << "\nSubject : " << Subject ;
    cout << "\nBody : " << Body << endl;
}

void SendPhone( string TextMessage)
{
    cout << "\nThe following SMS sent successfully to phone : " <<
_Phone ;
    cout << "\n" << TextMessage << endl;
}
};

int main()
{
    clsPerson Person1(10, "Saeed", "Omar", "My@gmail.com", "057842937");

    Person1.Print();

    Person1.SendEmail("Hi", "How are you ? ");
    Person1.SendPhone("How are you ? ");

    system("pause>0");
    return 0;
}

```

#Project 2: Homework : Employee Exercise

: Info

ID : 10

FirstName : Saeed

LastName : Omar

FullName : Saeed Omar

Title : Riyadh

Email : MySaeed@gmail.com

Phone : 05784732837

Salary : 2000

Department : Employee

The following message sent successfully to email : MySaeed@gmail.com

Subject : Hi

? Body : How are you

The following SMS sent successfully to phone : 05784732837

? How are you

```
#include <iostream>
using namespace std;
class clsEmployee
{
private:
    int _ID;
    string _FirstName;
    string _LastName;
    string _Title;
    string _Email;
    string _Phone;
```

```

    float _Salary;
    string _Department;

public:
    clsEmployee(int ID, string FirstName, string LastName, string Title,
string Email, string Phone, float Salary, string Department)
    {
        _ID = ID;
        _FirstName = FirstName;
        _LastName = LastName;
        _Title = Title;
        _Email = Email;
        _Phone = Phone;
        _Salary = Salary;
        _Department = Department;
    }
    int ID()
    {
        return _ID;
    }
    void setFirstName(int FirstName)
    {
        _FirstName = FirstName;
    }
    string FirstName()
    {
        return _FirstName;
    }
    void setLastName(string LastName)
    {
        _LastName = LastName;
    }
    string LastName()
    {
        return _LastName;
    }

    void setTitle(string Title)
    {
        _Title = Title;
    }
    string Title()
    {
        return _Title;
    }

    void setEmail(string Email)
    {
        _Email = Email;
    }
    string Email()
    {
        return _Email;
    }

    void setPhone(string Phone)
    {
        _Phone = Phone;
    }
    string Phone()
    {
        return _Phone;
    }

```

```

void setSalary(float Salary)
{
    _Salary = Salary;
}
float Salary()
{
    return _Salary;
}

void setDepartment(string Department)
{
    _Department = Department;
}
string Department()
{
    return _Department;
}

string FullName()
{
    return _FirstName + " " + _LastName;
}

void Print()
{
    cout << "\nInfo : ";
    cout << "\n-----";
    cout << "\nID : " << _ID;
    cout << "\nFirstName : " << _FirstName;
    cout << "\nLastName : " << _LastName;
    cout << "\nFullName : " << FullName();
    cout << "\nTitle : " << _Title;
    cout << "\nEmail : " << _Email;
    cout << "\nPhone : " << _Phone;
    cout << "\nSalary : " << _Salary;
    cout << "\nDepartment : " << _Department;
    cout << "\n-----" << endl;
}

void SendEmail(string Subject, string Body)
{
    cout << "\nThe following message sent successfully to email : " <<
_Email;
    cout << "\nSubject : " << Subject;
    cout << "\nBody : " << Body << endl;
}

void SendPhone(string TextMessage)
{
    cout << "\nThe following SMS sent successfully to phone : " <<
_Phone;
    cout << "\n" << TextMessage << endl;
}
};

int main()
{
    clsEmployee Employee1(10, "Saeed", "Omar", "Riyadh", "MySaeed@gmail.com",
"05784732837", 2000, "Employee");
    Employee1.Print();

    Employee1.SendEmail("Hi" , "How are you ?");
    Employee1.SendPhone( "How are you ?");
    system("pause>0");
    return 0;
}

```

#Lesson 20 : Third Principle / Concept of OOP : Inheritance

Inheritance : الوراثة (مهم جدا + فهمه بشكل جيد + كثرة الممارسة == سرعة مضاعفة في كتابة الكود)

في مشروع Person يوجد كثير من **Members** التي سيتم استخدامها في مشروع Employee مثل (ID , FirstName , Email) ، باستخدام مبدأ الوراثة **Inheritance** تستطيع نقل كل **Members** الخاصة بـ Person ويسمى (**Super Class / Base Class**) الى Employee ويسمى (**Sub Class / Derived Class**) : بسطر كود واحد ثم تضيف **Members** المختلفة في داخل class Employee

Inheritance تجعلك ترث الكود من غير نسخه ، أي تعديل أو إضافة **Super Class** يرثه **Sub Class**

Inheritance تتم وراثته (**Public & Protected Members**) فقط من **Super Class**

من فوائد استخدام **Inheritance** : توفير كتابة الكود == توفير وقت

- ❖ المحافظة على الكود Maintaining
- ❖ تحديث الكود Updating
- ❖ إصلاح الأخطاء أو التعديل – يكون في مكان واحد –
- ❖ إعادة استخدام الكود

```
#include <iostream>
using namespace std;
class clsPerson
{
    // private حتى لو لم تكتب فهي private
private:
    int _ID;
    string _FirstName;
    string _LastName;
    string _Email;
    string _Phone;
public:
    // Constructor إنشاء أكثر
    clsPerson()
    {
    }
    // لا تستطيع إنشاء Object باستخدام Inheritance سيتم حل المشكلة في درس لاحق
    // ما بين الأقواس يسمى ( Parameterize Constructor )
    clsPerson(int ID, string FirstName, string LastName , string Email ,string
Phone)
    {
        // Constructor هو أول Function يتم استدعاؤه عند إنشاء Object
        // Initialization تعبئة Object بالبيانات قبل استخدام Members
        _ID = ID;
        _FirstName = FirstName;
        _LastName = LastName;
        _Email = Email;
```

```

        _Phone = Phone;
    }
    // Lesson #19 Project2 : بقية الكود موجود في
}
class clsEmployee : public clsPerson
{
    // clsPerson يتم كتابة الأشياء المختلفة أو التي لا توجد في
    // clsEmployee أو كتابة الأشياء التي لها علاقة في
private:
    string _Title;
    string _Department;
    float _Salary;
public:
    //Property Set
    void setTitle(string Title)
    {
        _Title = Title;
    }
    //Property Get
    string Title()
    {
        return _Title;
    }

    //Property Set
    void setDepartment(string Department)
    {
        _Department = Department;
    }
    //Property Get
    string Department()
    {
        return _Department;
    }

    //Property Set
    void setSalary(float Salary)
    {
        _Salary = Salary;
    }
    //Property Get
    float Salary()
    {
        return _Salary;
    }
};
int main()
{
    // Lesson #20

    clsEmployee Employee1;
    Employee1.setFirstName("Mohammed");
    Employee1.setLastName("Abu-Hadhoud");
    Employee1.setEmail("a@a.com");
    Employee1.Print();
    Employee1.SendEmail("Hi", "How are you?");
    Employee1.setSalary(5000);
    cout << "Salary is: " << Employee1.Salary();

    //Calling the print will not print anything from derived class, only base class
    //therefore, the print method will not serve me here, this is a problem will be
    //solved in the next lecture.
    Employee1.Print();
}

```


// لن يؤدي استدعاء الطباعة إلى طباعة أي شيء من الفئة المشتقة، بل من الفئة الأساسية فقط
 // لذلك لن تخدمني طريقة الطباعة هنا، سيتم حل هذه المشكلة في المحاضرة القادمة

```
    system("pause>0");
    return 0;
}
```

Answers	Question	Quiz
True	Inheritance: Inheritance is one in which a new class is created that inherits the properties of the already exist class. It supports the concept of code reusability and reduces the length of the code in object-oriented programming	
True	The class that inherits properties from another class is called Subclass or Derived Class	
True	The class whose properties are inherited by a subclass is called Base Class or Superclass	
True	Derived Class and Sub Class are the same	
True	Base Class and Super Class are the same	
True	You can inherit only public and protected members; private members are not inherited	

#Lesson 21 : Parameterized Constructor of the Base Class

في الدرس السابق كانت هناك مشكلة وهي : لا تستطيع إنشاء **Object** من **Derived Class** التي ترث **Inheritance** من **Base Class** إلا بعد تعبئة **Parameterized** الخاصة بـ **Base Class**

قاعدة : لا تجعل أي أحد ينشئ **Object** فارغ من **Class** (أي بدون أن تكون فيه بيانات)

```
#include <iostream>
using namespace std;
class clsPerson
{
    // Lesson Project1 Person : #19 ارجع الى الكود في درس
}
// Inheritance ل Syntax هذا
// Sub Class / Derived Class تسمى clsEmployee
// Super Class / Base Class تسمى clsPerson
// هذه public أو غيرها ستدرس لاحقا
class clsEmployee : public clsPerson
{
    // clsPerson يتم كتابة الأشياء المختلفة أو التي لا توجد في
    // clsEmployee أو كتابة الأشياء التي لها علاقة في
private:
    // Lesson Project2 Employee : #19 ارجع الى الكود في درس
    string _Title;
    string _Department;
    float _Salary;
public:
    // إنشاء Constructor ل clsEmployee ووضع له Parameterized
    // تنادي clsPerson : وتضع قبله نقطتان رأسيتان ثم وضع Parameterized
    // هذا Constructor له مهمتان أخذ Parameterized ل 1. لكل ما يحتاجه clsEmployee
    // و 2. وما يحتاج له clsPerson
    clsEmployee(int ID, string FirstName, string LastName, string Email,
string Phone, string Title, string Department, float Salary)
        : clsPerson(ID, FirstName, LastName, Email, Phone)
    {
        _Title = Title;
        _Department = Department;
        _Salary = Salary;
    }
    // Lesson Inheritance: #20 ارجع الى الكود في درس
}
int main()
{
    // أول خمس باراميتر تتم مشاركتها مع clsPerson ( Base Class )
    clsEmployee Employee1 (10, "Mohammed", "Abu-Hadhoud", "A@a.com",
"8298982", "CEO", "ProgrammingAdvices", 5000);

    // كيف تهدم Function لا تحتاجها في Base Class وتبني مكانها في Function في Derived Class
    // يطبع فقط الأشياء الخاصة بـ clsPerson ( Base Class )
    Employee1.Print();

    cout << "\n" << Employee1.Title() << endl;
    cout << "\n" << Employee1.Department() << endl;
    cout << "\n" << Employee1.Salary() << endl;

    system("pause>0");
    return 0; }
```

#Lesson 22 : Function Overriding

كيف تتجاوز **Function Overriding** التي لا تحتاجها في **Base Class** وتبني مكانها **Function** في **Derived Class** بنفس الاسم – فتلغى **Base Class** ويحل محلها التي في **Derived Class** –

```
#include <iostream>
using namespace std;
class clsPerson
{
    // ارجع الى الكود في درس #19 Lesson Project1 Person
}
// Inheritance ل Syntax هذا
// Sub Class / Derived Class تسمى clsEmployee
// Super Class / Base Class تسمى clsPerson
// هذه public أو غيرها ستدرس لاحقا
class clsEmployee : public clsPerson
{
    // ارجع الى الكود في درس #20 Lesson Inheritance:

    // عند تسمية Function في Derived Class بنفس الاسم الموجود في Base Class
    // يتم مناداة Function في Derived Class ويتجاوز الآخر
    void Print()
    {
        // لا تستطيع الوصول بكل مباشر الى private الخاص ب Base Class
        // cout << "\nID : " << _ID;

        // وإنما تستطيع الوصول إليها عن طريق
        cout << "\nInfo:";
        cout << "\n-----";
        cout << "\nID : " << ID();
        cout << "\nFirstName : " << FirstName();
        cout << "\nLastName : " << LastName();
        cout << "\nFull Name : " << FullName();
        cout << "\nEmail : " << Email();
        cout << "\nPhone : " << Phone();

        cout << "\nTitle : " << _Title;
        cout << "\nDepartment: " << _Department;
        cout << "\nSalary : " << _Salary;
        cout << "\n-----\n";

        // تستطيع الوصول واستخدام Function الموجود في Base Class
        clsPerson::Print();
    }
}

int main()
{
    clsEmployee Employee1 (10, "Mohammed", "Abu-Hadhoud", "A@a.com",
    "8298982", "CEO", "ProgrammingAdvices", 5000);

    // يتم استدعاء Print(); الخاصة ب clsEmployee
    Employee1.Print();

    system("pause>0");
    return 0;
}
```

Answers	Question	Quiz
True	Function Overriding : The function in derived class overrides the function in base class	
False	If you override a function in base class will you be able to access this function from the object of derived class	
True	If you override a function in base class will you be able to access this function inside derived class	
BaseClass::FunctionName()	If you override a function in base how to access it from within the derived class	

#Lesson 23 : Multi Level Inheritance

يكون **Derived Class** Developer يرث **Base Class** Employee
و **Derived Class** Employee يرث **Base Class** Person

Last Homework - Developer



Super Class/Base Class

Person:
ID
First name
Last name
FullName()
Email
Phone
SendEmail(..)
SendSMS(...)
Print()

Sub Class/Derived Class
Super Class/Base Class

Employee:
ID
First name
Last name
FullName()
Title
Email
Phone
Salary
Department
SendEmail(..)
SendSMS(...)
Print()

Inherits

Inherits

Sub Class/Derived

Developer:
ID
First name
Last name
FullName()
Title
Email
Phone
Salary
Department
SendEmail(..)
SendSMS(...)
MainProgrammingLanguage
Print()



Copyright© 2022

Mohammed Abu-Hadoud
MBA, PMOC, PgMP®, PRINCE2®, PMP®, PMI-ACP®, CS, ITIL, MCP®, MCSD
The power of knowledge

```
#include <iostream>
using namespace std;
class clsPerson
{
    // Lesson Project1 Person : #19 ارجع الى الكود في درس
}
class clsEmployee : public clsPerson
{
    // Lesson Function Overriding: #22 ارجع الى الكود في درس
}
```

```

class clsDeveloper : public clsEmployee
{
private:
    string _MainProgrammingLanguage;

public:
    clsDeveloper(int ID, string FirstName, string LastName, string Email,
string Phone,
        string Title, string Department, float Salary, string
MainProgrammingLanguage)
        : clsEmployee(ID, FirstName, LastName, Email, Phone, Title,
Department, Salary)
    {
        _MainProgrammingLanguage = MainProgrammingLanguage;
    }
    //Property Set
    void setMainProgrammingLanguage(string MainProgrammingLanguage)
    {
        _MainProgrammingLanguage = MainProgrammingLanguage;
    }
    // Property Get
    string MainProgrammingLanguage()
    {
        return _MainProgrammingLanguage;
    }

    // Function Overriding
    void Print()
    {
        cout << "\nInfo:";
        cout << "\n-----";
        cout << "\nID" : " << clsPerson::ID();
        cout << "\nFirstName" : " << clsPerson::FirstName();
        cout << "\nLastName" : " << clsPerson::LastName();
        cout << "\nFull Name" : " << clsPerson::FullName();
        cout << "\nEmail" : " << clsPerson::Email();
        cout << "\nPhone" : " << clsPerson::Phone();

        cout << "\nTitle" : " << clsEmployee::Title();
        cout << "\nDepartment" : " <<
clsEmployee::Department();
        cout << "\nSalary" : " << clsEmployee::Salary();

        cout << "\nMain Programming Language : " <<
_MainProgrammingLanguage;
        cout << "\n-----\n";
    }
};

int main()
{
    clsDeveloper Developer1(10, "Mohammed", "Abu-Hadhoud", "A@a.com",
"8298982", "Web Developer", "ProgrammingAdvices", 5000, "C++");

    Developer1.Print();
    Developer1.SendPhone("Hi me Developer :-)");

    system("pause>0");
    return 0;
}

```

: Access Specifiers (Modifiers)

يطبق على كل شيء سواء Method & Property أو Variable & Function

يوجد ثلاثة أنواع من Access Specifiers (Modifiers) لـ Members
بمعنى آخر من الذي له الصلاحية للوصول إلى Members

١. **Private** : تستطيع الوصول إلى Members من داخل Class فقط
٢. **Protected** : يكون داخل Class فقط + كل Classes التي ترث Inheritance هذه Class
٣. **Public** : أي أحد يستطيع الوصول إلى Members من أي مكان

Private هو للاستخدام الداخلي لـ Class فقط أو لحماية Members من التعديل عليه من الخارج

```
#include <iostream>
using namespace std;
class clsC
{
private:
    //only accessible inside this class, neither derived classes nor outside
class
    int _Var1;
    void _Fun1()
    {
        cout << "Function 1";
    }

protected:
    //only accessible inside this class and all derived classes, but not
outside class
    int Var2;
    void Fun2()
    {
        cout << "Function 2";
    }

public:
    // Accessible inside this class, all derived classes, and outside class
    int Var3;
    void Fun3()
    {
        cout << "Function 3";
    }
};

class clsD : public clsC
{
public:
    void Func1()
    {
        // Base Class من Inheritance التي ترث Class من داخل Class protected + public Members
        // تستطيع رؤية
        cout << clsC::Var2;
    }
}
```

```
};  
  
int main()  
{  
    // في Object. تستطيع رؤية Public Members فقط  
  
    clsC C;  
    C.Fun3();  
    C.Var3;  
  
    clsD D;  
    D.Fun3(); // clsC  
    D.Func1(); // clsD  
    D.Var3; // clsC  
  
    system("pause>0");  
    return 0;  
}
```


Answers	Question	Quiz
True	Access modifiers (or access specifiers) are keywords in object-oriented languages that set the accessibility of classes, methods, and other members	
Private .١ Protected .٢ Public .٣	Which of the following is Access Specifiers/Modifiers	
True	Public Members can be accessed from inside and outside the class	
False	Private Members can be accessed from outside the class through object	
False	Private Members can be accessed by any class inherits the current class	
True	Private Members can be accessed only from inside the class, it cannot be accessed from outside the class nor from the classes inherits the current class	
Protected	If you want to have a member that is private to outside class and public to classes inherits the current class, which access specifier/modifier you use	
False	Protected Members can be accessed from outside class through objects	
True	Protected Members can be accessed from inside class and from all classes inherits the current class	
True	OOP is more secured because you can hide members from developers	
True	Inside the class I can access everything including Public, Private , and Protected Members	

#Lesson 25 : Inheritance Visibility Modes

// هذه public أو غيرها ستدرس في هذا الدرس
`class clsEmployee : public clsPerson`

```
class DerivedClassName : <Visibility Mode> BaseClassName
{
}
```

public : تجعل كل من يرث من `DerivedClassName` يستطيع الوصول الى كل **Members** التي ورثها من `BaseClassName` إلا **Private**

مثال `class DerivedClassName : public BaseClassName` (يرث من **Base** : **Members** التالية)

- **Private Member** : لا أحد يستطيع الوصول إليها من خارج `BaseClassName`
- **Protected Member** : تستطيع الوصول إليها
 - من داخل `DerivedClassName`
 - ومن يرث من `DerivedClassName`
- **Public Member** : تستطيع الوصول إليها من
 - داخل `DerivedClassName`
 - ومن يرث من `DerivedClassName`
 - ومن **Object**

private : تجعل كل من يرث من `DerivedClassName` لا يستطيع الوصول الى أي **Members** تمت وراثتها من `BaseClassName` (تجعله خاص **Private** فقط بهذه `DerivedClassName`)

مثال `class DerivedClassName : private BaseClassName` (يرث من **Base** : **Members** التالية)

- **Private Member** : لا أحد يستطيع الوصول إليها من خارج `BaseClassName`
- **Protected Member** : تستطيع الوصول إليها من داخل هذه `DerivedClassName` فقط
- **Public Member** : تستطيع الوصول إليها من داخل `DerivedClassName` فقط

يعني أن كل **Members** (**Public & Protected**) يصحان **Private** ل `DerivedClassName` لا يستطيع الوصول الى **Members** الموروثة إلا من داخل `DerivedClassName` فقط لا يتم توريث **Members** التي تمت وراثتها من `BaseClassName` لأي **Class** آخر

protected : تجعل كل من يرث من `DerivedClassName` يستطيع الوصول الى **Members** التي ورثها من `BaseClassName` (**مهما تعددت** الوراثة ممن يرث من `DerivedClassName`)

مثال `protected BaseClassName` : `class DerivedClassName` (يرث من `Base` : `Members` التالية)

- **Private Member** : لا أحد يستطيع الوصول إليها من خارج `BaseClassName`
- **Protected Member** : يستطيع الوصول إليها
 - من داخل `DerivedClassName`
 - ومن يرث من `DerivedClassName2`
 - ومن يرث ممن يرث من `DerivedClassName2`
- **Public Member** : يستطيع الوصول إليها من
 - داخل `DerivedClassName`
 - ومن يرث من `DerivedClassName2`
 - ومن يرث ممن يرث من `DerivedClassName2`

يعني أن كل **Members** (`Public & Protected`) يصبحان `Protected` ل `DerivedClassName` ، لا يستطيع الوصول إليها إلا من داخل `Class` يرث من `DerivedClass`

Visibility Modes	Private Members	Protected Members	Public Members
Public Inheritance	Inaccessible	Protected	Public
Private Inheritance	Inaccessible	Private	Private
Protected Inheritance	Inaccessible	Protected	Protected

عادة يتم استخدام **public** في Inheritance

```

#include <iostream>
using namespace std;

class clsA
{
private:
    int V1;

    int Fun1()
    {
        return 1;
    }

protected:
    int V2;

    int Fun2()
    {
        return 2;
    }

public:
    int V3;

    int Fun3()
    {
        return 3;
    }
};

//try to change visibility mode public/private/protected
//and see in the main what will happen inside objects.
// clsB يستخدم في داخل clsA فقط
class clsB : private clsA
{
public:
    int Fun4()
    {
        // clsB من داخل public & protected الوصول الى
        clsA::Fun2();
        clsA::V2;

        clsA::Fun3();
        clsA::V3;

        return 4;
    }
};

class clsC : public clsB
{
public:
    int Fun5()
    {
        clsB::Fun4();
        // Object طريق
        clsB B1;
        B1.Fun4();

        return 5;
    }
};

```

// clsD من داخل أي Class وراث من clsA
// أو من يرث عن يرث من clsD

```
class clsD : protected clsA
```

```
{
```

```
public:
```

```
    int Fun6()
```

```
    {
```

```
        clsA::Fun2();
```

```
        clsA::V2;
```

```
        clsA::Fun3();
```

```
        clsA::V3;
```

```
        return 6;
```

```
    }
```

```
};
```

```
class clsE : protected clsD
```

```
{
```

```
public:
```

```
    int Fun7()
```

```
    {
```

```
        clsD::Fun2();
```

```
        clsD::V2;
```

```
        clsD::Fun3();
```

```
        clsD::V3;
```

```
        clsD::Fun6();
```

```
        return 7;
```

```
    }
```

```
};
```

```
class clsF : public clsE
```

```
{
```

```
public:
```

```
    int Fun8()
```

```
    {
```

```
        clsE::Fun2();
```

```
        clsE::V2;
```

```
        clsE::Fun3();
```

```
        clsE::V3;
```

```
        clsE::Fun6();
```

```
        clsE::Fun7();
```

```
        return 8;
```

```
    }
```

```
};
```

```
int main()
```

```
{
```

// Object من Public Members الوصول الى كل

```
clsA A1;
```

```
A1.Fun3();
```

```
A1.V3;
```

```

//تستطيع الوصول الى كل Public Members من Object + أي BaseClass تم وراثتها بشكل public
//ولا تستطيع الوصول الى أي شيء تمت وراثته بشكل private من Base Class في Object
    clsB B1;
    B1.Fun4();
//Try B1 after you change visibility mode in clsB.    and see what you can see.

    //تستطيع الوصول الى كل Public Members من Object
    //ولا تستطيع الوصول الى أي شيء تمت وراثته بشكل private من Base Class
    clsC C1;
    C1.Fun4();
    C1.Fun5();
//Try C1 after you change visibility mode in clsB.    and see what you can see

    //تستطيع الوصول الى كل Public Members من Object
    //ولا تستطيع الوصول الى أي شيء تمت وراثته بشكل private من Base Class
    clsD D1;
    D1.Fun6();
//Try D1 after you change visibility mode in clsB.    and see what you can see

    //تستطيع الوصول الى كل Public Members من Object
    //ولا تستطيع الوصول الى أي شيء تمت وراثته بشكل private & protected من Base Class
    clsE E1;
    E1.Fun7();
//Try E1 after you change visibility mode in clsB.    and see what you can see

    clsF F1;
    F1.Fun7();
    F1.Fun8();

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	Visibility Mode "Private" will make everything private in the derived class, so it can make use of it and no one will make use of it neither derived classes nor objects	
True	Visibility Mode "Protected" will make everything protected in the derived class, so it can make use of it and all other derived classes will make use of it, but no objects will make use of it	
True	Visibility Mode "Public" will inherit the class publicly so every public members and protected members are useful for others	
True	Private Members in the Base Class are not accessible from outside the class nor the derived classes	

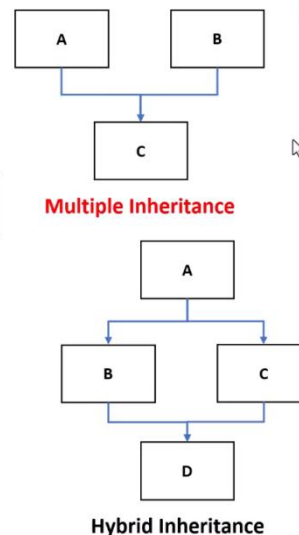
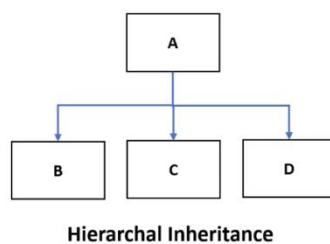
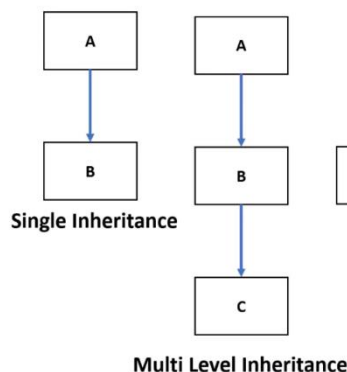
أنواع Inheritance (هؤلاء 3 مستخدمين في أغلب لغات البرمجة)

١. **Single Inheritance** مثال `clsB` يرث من `clsA`
٢. **Multi-Level Inheritance** مثال `clsC` يرث من `clsB` وهو يرث من `clsA`
٣. **Hierarchal Inheritance** مثال `clsD` و `clsC` و `clsB` الكل يرث من `clsA`

أنواع Inheritance الإضافية الموجودة فقط في C++ (استخدامه قد يسبب مشاكل في البرنامج)

- ☒ **Multiple Inheritance** مثال `clsC` يرث من - اثنان - `clsA` و `clsB`
- ☒ **Hybrid Inheritance** مثال `clsD` يرث من - اثنان - `clsB` و `clsC` وهما يرثان من `clsA`

Inheritance Types



Answers	Question	Quiz
(Single & Multi-Level & Hierarchal) (Multiple & Hybrid) Inheritance	What are types of Inheritance ?	
True	Multiple inheritance are not supported by modern languages such as JAVA and C#	

#Lesson 27 : Up Casting vs Down Casting

Casting هو : تحويل نوع Type الى نوع Type آخر مثل التحويل من int الى double

في OOP بالأخص في الوراثة **Inheritance** يوجد نوعين من Casting

١. **Up Casting** هو التحويل من **Derived Class** الى **Base Class**

(التحويل من الأكبر الى الأصغر) لأن Members في **Base** موجودة في **Derived**

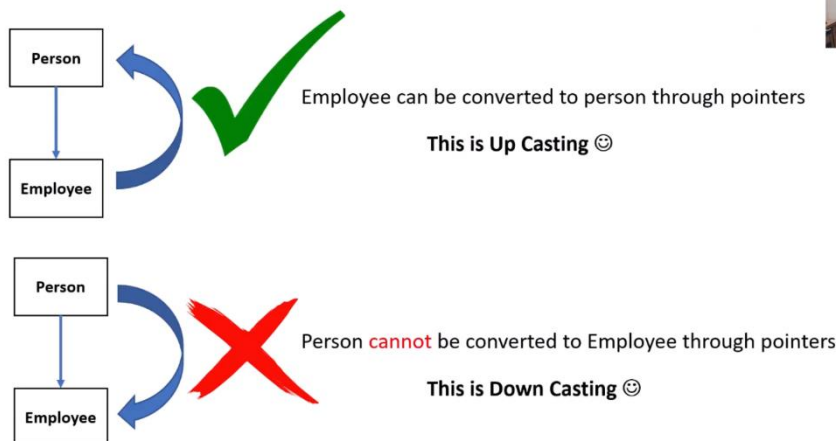
a. عن طريق Pointers : Object من نوع **Base** يؤشر على Object من نوع **Derived**

b. **Base Pointers * Object** تأخذ Members من **Derived** التي ورثها من **Base**

٢. **Down Casting** هو التحويل من **Base Class** الى **Derived Class** عن طريق Pointers

(التحويل من الصغير الى الكبير) لأنه توجد Members في **Derived** زائدة عن **Base**

Up Casting vs Down Casting



```
#include <iostream>
using namespace std;
```

```
class clsPerson
{
public:
    string FullName = "Mohammed Abu-Hadhoud";
};

class clsEmployee: public clsPerson
{
public:
    string Title = "CEO";
};
```



```

int main()
{
    clsEmployee Employee1;

    cout << Employee1.FullName << endl;

    ///upcasting
    //this will convert employee to person.
    // Address يتم تحويل & Employee1
    // clsPerson يأخذ من Employee1 الأشياء Members التي تم وراثتها من
    clsPerson * Person1 = &Employee1;
    cout << Person1->FullName << endl;

    // clsEmployee لا تستطيع الوصول الى Members الخاصة ب
    // clsPerson لأن Person1 يستطيع الوصول Members فقط الخاصة ب
    //cout << Person1->Title << endl;

    //clsPerson Person2;
    //cout << Person2.FullName << endl;
    ///down casting : you cannot convert person to employee
    /// clsEmployee لأن Person2 لا يوجد فيه Members تشمل Members الخاصة ب
    //clsEmployee* Employee2 = &Person2;

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	Up Casting is converting derived object to it's base object	
True	Down Casting is Converting Base object to Derived object	
True, because all the members in which the pointer can access are exist in memory when the object of child class	A pointer of type parent can point to an object of child class	
True, because the child class members the pointer can access do not exist in memory when the object is of parent class	A pointer of child class cannot point to an object of parent class	

#Lesson 28 : Virtual Functions

عند استخدام **Overriding** ... ثم تستعمل **upcasting** فتحدث مشكلة استدعاء **Method** المطلوبة ، فيستدعي **Method** الخاصة في Base بدل **Method** الخاصة في Derived وحل هذه المشكلة هي : إضافة كلمة **Virtual** قبل تعريف أو إنشاء **Method** في Base

Virtual Functions يتم استخدامه مرة واحدة ل **Overriding**

تنبه Compiler أنه سيتم استخدامه في الوراثة **Overriding** ، وأيضا في Pointers سيتم إنشاء مكان في الذاكرة اسمه **Virtual table** يتم تخزين فيه كل Address الخاصة ب **Overriding Method** لكيلا يحدث خطأ في الاستدعاء

استخدام **Virtual** تبطئ البرنامج بشكل قليل لأنه يحجز مكان في الذاكرة

إذا استخدمت **Overriding** استخدم معها **Virtual**

```
#include <iostream>
using namespace std;
class clsPerson
{
public:
    virtual void Print()
    {
        cout << "Hi, I'm a person!\n ";
    }
};

class clsEmployee: public clsPerson
{
public:
    // Function Overriding
    void Print()
    {
        cout << "Hi, I'm an Employee\n";
    }
};

class clsStudent: public clsPerson
{
public:
    // Function Overriding
    void Print()
    {
        cout << "Hi, I'm a student\n";
    }
};

int main()
{
```

```

    clsEmployee Employee1;
    clsStudent Student1;

    Employee1.Print();
    Student1.Print();

    // upcasting
    clsPerson * Person1 = &Employee1;
    clsPerson * Person2 = &Student1;

    Person1->Print();
    Person2->Print();

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	A virtual function is a member function in the base class that we expect to redefine in derived classes	
True	Basically, a virtual function is used in the base class in order to ensure that the function is overridden. This especially applies to cases where a pointer of base class points to an object of a derived class	

#Lesson 29 : (Static / Early) Binding vs (Dynamic / Late) Binding

(**Static / Early**) Binding تم ربط Address قبل تشغيل البرنامج ب Member التابع ل Object الخاص Class

(**Dynamic / Late**) Binding يتم ربط Address وقت تشغيل البرنامج ب Virtual Function التابع ل Base Class المعروف عن طريق Pointers

بمعنى آخر تم ربط كل Address ب Member & Method قبل تشغيل البرنامج إلا Pointers & Virtual Function يتم ربطها أثناء تشغيل البرنامج

لذا (**Static / Early**) Binding أسرع من (**Dynamic / Late**) Binding

```
#include <iostream>
using namespace std;
class clsPerson
{
public:
    virtual void Print()
    {
        cout << "Hi, I'm a person!\n ";
    }
};

class clsEmployee: public clsPerson
{
public:
    // Function Overriding
    void Print()
    {
        cout << "Hi, I'm an Employee\n";
    }
};

class clsStudent: public clsPerson
{
public:
    // Function Overriding
    void Print()
    {
        cout << "Hi, I'm a student\n";
    }
};
```

```

int main()
{
    clsEmployee Employee1;
    clsStudent Student1;

    // Early - Static Binding : at compilation time      Employee1.Print();
    Student1.Print();

    // upcasting
    //Late-Dynamic Binding: at runtime
    clsPerson * Person1 = &Employee1;
    clsPerson * Person2 = &Student1;

    Person1->Print();
    Person2->Print();

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	Static Binding: The binding which can be resolved at compile time by the compiler is known as static or early binding. The binding of all the static, private methods is done at compile-time	
True	Dynamic Binding: In Dynamic binding compiler doesn't decide the method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have the same method	
True	Early Binding and Static Binding are the same	
True	Late Binding and Dynamic Binding are the same	
True	Early Binding / Static Binding are done at the compilation time	
True	Late Binding/Dynamic Binding are done at run time	

#Lesson 30: Fourth (Principle / Concept) of OOP : Polymorphism

Polymorphism يعني تعدد الأشكال + إنشاء كود مرتب وعلى نسق واحد + يخفف الأخطاء على Developer + سهولة تذكّر اسم Function

يسمح **Polymorphism** للكود بالتعامل في **Object** مع أنواع مختلفة دون الحاجة إلى كتابة كود مختلف لكل نوع

يمكن تحقيق Polymorphism عن طريق

١. **Function Overloading** مثال : يكون عندك أكثر من Function لهم اسم واحد ولكنهم

يختلفون في عدد أو نوع Parameters (يتم استدعاؤهم باسم واحد)

٢. **Operator Overloading** العوامل وهي / , * , - , + وغيرها مثال + : لجمع عددين أو

إلصاق نصين بجانب بعضهما

٣. **Function Overriding** مثال **Derived Class** ترث من **Base Class** ويكون لديهما

نفس اسم Function لكن يتم إلغاء أو تجاوز Function في **Base Class** إلى Function

الخاص ب **Derived Class**

٤. **Virtual Functions** وتسمى **Run Time Polymorphism**

ولها علاقة **Function Overriding** مثال : **Object** من نوع **Base Class** يؤشر على

Derived Class فيصل من خلاله إلى Function الخاص ب **Derived**

Answers	Question	Quiz
True	Why Polymorphism? Polymorphism allows us to create consistent code	
True	Polymorphism is one of the important features/principles/concepts of OOP, word Ploy means "Many" and word Morphism means "Form" so it means "Many Forms", the ability to take more than one form	
Function Overloading Operator Overloading Function Overriding Virtual Methods	We can achieve Polymorphism through	

إنشاء Class من نوع عقد **Interface أو Contract**

إجبار Developer عند إنشائه **Derived Class** التي ترث من **Base Class** على إنشاء كل **Pure Virtual Function** **Certain (Method / Parameters)** التي تتواجد في **Base** بشكل وهذا يسمى : **Interfaces أو Abstract Classes أو Contract**

عند إنشاء **Pure Virtual Function** واحد في **Class** فقد جعلتها بمثابة العقد **Contract أو Interface** ويترتب عليها ما يلي – منها –

- لا تستطيع إنشاء **Object** منها
- إلزام من يرث هذه **Class** على وجود كل **(Method/Parameters)** المذكورة في **Contract**
- تعطيك تحكم في الكود

هناك فرق بين Abstract Classes وبين مبدأ أو مفهوم Abstraction

```
#include <iostream>
using namespace std;
//Abstract Class / Interface / Contract.
class clsMobile
{
    // تحول هذا Function الى Pure Virtual
    // Function من نوع virtual له واجهة فقط Interface وفي الأخير = 0
    // يعني لم نجعلها { Implementation } بداخلها أوامر
    // لذا لا تستطيع إنشاء Object منها
    // بمجرد وجود Pure Virtual تحولت هذه Class الى
    //Abstract Class / Interface / Contract.

    virtual void Dial(string PhoneNumber) = 0;
    virtual void SendSMS(string PhoneNumber, string Text) = 0;
    virtual void TakePicture() = 0;
};

class clsiPhone : public clsMobile
{
    //This class signed a contract with clsMobile abstract class therefore it
    //should implement everything in the abstract class.
    // Pure Virtual clsMobile يعني أنه شرط عليك تنفيذ كل شروطه وهي
    public:
        void Dial(string PhoneNumber)
        {
            // { Implementation }; لابد من القوسين ثم فاصلة منقوطة
        };

        void SendSMS(string PhoneNumber, string Text)
        {
        };
};
```

```

void TakePicture()
{
};

// Contract إنشاء Method زائدة عن العقد
void MyOwnMethod()
{
}

};

class clsSamsungNote10: public clsMobile
{
    //This class signed a contract with clsMobile abstract class therefore
    it should implement everything in the abstract class
public:
    void Dial(string PhoneNumber)
    {
    };

    // clsSamsungNote10 من Object إنشاء لن تستطيع كل الشروط
    //void SendSMS( string Text)
    //{
    //};

    void SendSMS(string PhoneNumber, string Text)
    {
    };

    void TakePicture()
    {
    };
};

int main()
{
    clsiPhone iPhone1;
    clsSamsungNote10 Note10;

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	A pure virtual function doesn't have the function body and it must end with = 0	
True	If you have one pure virtual function in a class then it will be converted to abstract class	

True	Abstract Class is the same concept of Interface Class and they are both contracts
True	Abstract Class/Interface Class is a class with pure virtual functions
No, you can only inherit it	You can have an object of abstract class
True	An abstract class in C++ has at least one pure virtual function by definition. In other words, a function that has no definition
True	The abstract class's descendants (derived classes) must define the pure virtual function; otherwise, it is not allowed and you will get error
Yes, it can have extra methods	Derived Classes from abstract class can have extra methods other than the methods in the abstract class
True	The C++ interfaces are implemented using abstract classes and these abstract classes should not be confused with data abstraction which is a concept of keeping implementation details separate from associated data
No, they are two different things, these abstract classes should not be confused with data abstraction which is a concept of keeping implementation details separate from associated data	Abstract Class is the Same as Abstraction in OOP

#Lesson 32 : Friend Classes

قاعدة : لا تستطيع الوصول الى **Protected** أو الى **Private** عن طريق **Object**
أما **Public** تستطيع الوصول إليه من أي مكان – من داخل **Derived** أو من **Object**
أما **Protected** تستطيع الوصول إليها من داخل **Class** الذي ورث من **Base Class**
أما **Private** لا تستطيع الوصول إليها إلا من داخل **Class** نفسه

يوجد استثناء لهذه القاعدة وهي : مصاحبة **ClsA** وتستطيع أن تصل الى **Protected** أو الى **Private** أو الى **Public** من **ClsB** المصاحبة لـ **ClsA** لكن من داخل **ClsB** فقط
وكود المصاحبة يكتب في داخل **ClsA** وهو

```
friend class clsB;

#include <iostream>
using namespace std;
class clsA
{
private:
    int _Var1;

protected:
    int _Var3;

public:
    int Var2;

    clsA()
    {
        _Var1 = 10;
        Var2 = 20;
        _Var3 = 30;
    }

    //This will grant access for everything to class B
    friend class clsB; //friend class
};

class clsB
{
public:

    void display(clsA A1)
    {
        cout << endl << "The value of Var1=" << A1._Var1 ;
        cout << endl << "The value of Var2=" << A1.Var2 ;
        cout << endl << "The value of Var3=" << A1._Var3 ;
    }
};

int main()
{

    clsA A1;
```

```

    clsB B1;

    B1.display(A1);

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	We can use a friend Class in C++ using the "friend" keyword	
True	A friend class can access both private and protected members of the class in which it has been declared as friend	
True	Since ClassB is a friend class, we can access all members of ClassA from inside ClassB. However, we cannot access members of ClassB from inside ClassA. It is because friend relation in C++ is only granted, not taken	
False, only ClassB can access all members of ClassB but ClassA cannot	If ClassB is declared as a friend Class inside ClassA , ClassB can access all private and protected members of ClassA, and also ClassA can Access all members of ClassB	

#Lesson 33 : Friend Function

يمكن ل Function مصاحبة ClsA ويستطيع أن يصل الى **Protected** أو الى **Private** أو الى **Public** من Function المصاحبة ل ClsA لكن من داخل Function فقط وكود المصاحبة يكتب في داخل ClsA وهو

```
friend int MySum(clsA A1);
```

Friend Function : هو Function مكتوب خارج clsA – لا يتبع أي Class – ويمكن أن يكون داخل Class أخرى غير clsA ، وهذا Function يمكن أن يرجع قيمة – Return – أو لا يرجع قيمة

```
#include <iostream>
using namespace std;
class clsA
{
private:
    int _Var1;

protected:
    int _Var3;

public:
    int Var2;

    clsA()
    {
        _Var1 = 10;
        Var2 = 20;
        _Var3 = 30;
    }

    friend int MySum(clsA A1);    //friend function
};

//this function is a normal function and not a member of any class
int MySum(clsA A1)
{
    return A1._Var1 + A1.Var2 + A1._Var3 ;
}

int main()
{
    clsA A1;
    cout << MySum(A1);

    system("pause>0");
    return 0;
}
```

Answers	Question	Quiz
True	A friend function in C++ is defined as a function that can access private, protected and public members of a class	
True	The friend function is declared using the friend keyword inside the body of the class	
True	By using the keyword, the 'friend' compiler understands that the given function is a friend function	
True	We declare friend function inside the body of a class, whose private and protective data needs to be accessed, starting with the keyword friend to access the data. We use them when we need to operate between two different classes at the same time	
True	Friend functions of the class are granted permission to access private and protected members of the class in C++. They are defined globally outside the class scope. Friend functions are not member functions of the class	
True	A friend function in C++ is a function that is declared outside a class but is capable of accessing the private and protected members of the class. There could be situations in programming wherein we want two classes to share their members. These members may be data members, class functions or function templates. In such cases, we make the desired function, a friend to both these classes which will allow accessing private and protected data of members of the class	
True	Generally, non-member functions cannot access the private members of a particular class. Once declared as a friend function, the function is able to access the private and the protected members of these classes	

#Lesson 34 : Structure Inside Class

هل تستطيع إنشاء Structure داخل Class ؟

نعم ، لأنك تستطيع إنشاء : ... string , bool , int وكلها **Data Type** وكذلك Structure

```
#include <iostream>
using namespace std;
class clsPerson
{
    struct stAddress
    {
        string AddressLine1;
        string AddressLine2;
        string City;
        string Country;
    };
public:
    string FullName;
    stAddress Address;

    clsPerson()
    {
        FullName = "Mohammed Abu-Hadhoud";
        Address.AddressLine1 = "Building 10";
        Address.AddressLine2 = "Queen Rania Street";
        Address.City = "Amman";
        Address.Country = "Jordan";
    }

    void steCity(string City)
    {
        Address.City = City;
    }
    string City()
    {
        return Address.City;
    }
    void PrintAddress()
    {
        cout << "\nAddress:\n";
        cout << Address.AddressLine1 << endl;
        cout << Address.AddressLine2 << endl;
        cout << Address.City << endl;
        cout << Address.Country << endl;
    }
};

int main()
{
    clsPerson Person1;
    Person1.PrintAddress();

    Person1.steCity("Madinah");
    cout << "\n" << Person1.City() << endl;
    Person1.PrintAddress();

    system("pause>0");
    return 0;
}
```

#Lesson 35: Nested Classes

هل تستطيع إنشاء Class داخل Class أخرى ؟

نعم ، لأنك تستطيع إنشاء : ... string , Structure , int وكلها **Data Type** وكذلك Class

تعامل Class الداخلية كأى Class آخر

Inner Class : Class الداخلية تسمى :

Enclosing Class : Class المحتوية ل Class الداخلية تسمى :

```
#include <iostream>
using namespace std;
class clsPerson
{
    string _FullName;

    class clsAddress
    {
    private:
        string _AddressLine1;
        string _AddressLine2;
        string _City;
        string _Country;

    public:
        clsAddress (string AddressLine1, string AddressLine2, string City,
string Country)
        {
            _AddressLine1 = AddressLine1;
            _AddressLine2 = AddressLine2;
            _City = City;
            _Country = Country;
        }

        string setAddressLine1(string AddressLine1)
        {
            _AddressLine1 = AddressLine1;
        }
        string AddressLine1()
        {
            return _AddressLine1;
        }

        string setAddressLine2(string AddressLine2)
        {
            _AddressLine2 = AddressLine2;
        }

        string AddressLine2()
        {
            return _AddressLine2;
        }
    }
}
```

```

        string setCity(string City)
        {
            _City = City;
        }

        string City()
        {
            return _City;
        }

        string setCountry(string Country)
        {
            _Country = Country;
        }

        string Country()
        {
            return _Country;
        }

        void Print()
        {
            cout << "\nAddress:\n";
            cout << _AddressLine1 << endl;
            cout << _AddressLine2 << endl;
            cout << _City << endl;
            cout << _Country << endl;
        }
    };

public:
    string setFullName(string FullName)
    {
        _FullName = FullName;
    }
    string FullName()
    {
        return _FullName;
    }

    clsAddress Address = clsAddress("", "", "", "");

    clsPerson(string FullName, string AddressLine1, string AddressLine2,
string City, string Country)
    {
        _FullName = FullName; //initiate address class by it's constructor

        Address =clsAddress (AddressLine1, AddressLine2, City, Country);
    }

    void PrintInfo()
    {
        cout << "\nInfo : ";
        cout << "\n===== ";
        cout << "\nFull Name      : " << _FullName;
        cout << "\nAddress Line1 : " << Address.AddressLine1();
        cout << "\nAddress Line2 : " << Address.AddressLine2();
        cout << "\nCity          : " << Address.City();
        cout << "\nCountry       : " << Address.Country();
        cout << "\n===== " << endl;
    }
};

```



```

int main()
{
    clsPerson Person1("Mohammed Abu-Hadhoud", "Building 10", "Queen Rania
Street", "Amman", "Jordan");

    Person1.Address.Print();
    Person1.PrintInfo();

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	Nested or Inner Classes : A class can also contain another class definition inside itself, which is called “Inner Class” in C++	
True	In the case of nested or inner classes, the containing class is referred to as the “Enclosing Class”. The Inner Class definition is considered to be a member of the Enclosing Class	
True	An Inner class is a member and as such has the same access rights as any other member of the enclosure class	
True	The members of an enclosing class have no special access to members of a nested class; the usual access rules shall be obeyed	

#Lesson 36: Separate Classes In Libraries

كيف تنشأ مكتبة ل Class ؟ يتم إنشاء مكتبة لكل Class واحد

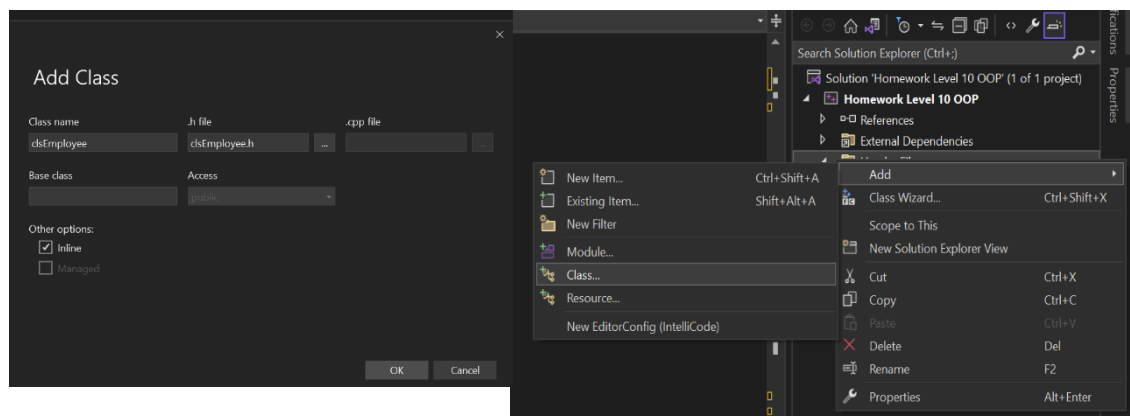
١. View

٢. Solution Explorer

٣. Header Files (زر الفأرة الأيمن)

٤. Add

٥. Class



- لا تستطيع إنشاء مكتبة ل Class و Class موجود في البرنامج (بنفس الاسم)
- لابد من إضافة المكتبات التي ستستخدمها داخل مكتبة Class – ولو كانت مكتبة Class أخرى –
- طريقة استدعاء المكتبة – داخلية – داخل البرنامج `#include "clsPerson.h"`
- **لا بد إضافة** `#pragma once` داخل أي مكتبة يتم إنشاؤها – وإلا سيحدث خطأ أثناء تشغيل البرنامج إذا تم استدعاء المكتبة في أكثر من مكان ك داخل البرنامج و مكتبة أخرى

Answers	Question	Quiz
True	Separating Code and Classes in Libraries will make our life easier and we can control our code and organize it better	
True	We must user "#pragma once" in each header file to prevent the compiler from loading the library more than one time and have repeated code included	

#Lesson 37 : What is 'this' Pointer ?

- ❖ Compiler يستخدم **'this' Pointer** بشكل افتراضي في داخل Class – لذا لا داعي لكتابتها
- ❖ عند إنشاء **Object** يتم إنشاء **Pointer** معه ويتم تخزين **Pointer** في **'this'**
- ❖ **'this'** هو **Pointer** للدلالة على أي **Object** من **Class**
- ❖ **'this'** تستخدم في داخل **Class** فقط
- ❖ من فوائد **'this'** : أنك تستطيع الوصول الى أي شيء في داخل **Class** عن طريق **'this'**
- ❖ **'this'** توصلك الى كل **Data Members** الخاصة ب **Object** من داخل **Class**
 - سواء كانت **Data Members** : **Public** , **Private** , **Protected**
 - ما عدا **friend functions** لأنها ليست **Member**
 - لأنها **Reference** ل **functions** خارجي
- ❖ **'this'** مهم جدا لأنك في حالات معينة لا تستطيع حل المشكلة إلا ب **'this'**
 - منها إنشاء **Variable** و **Parameter** بنفس الاسم
 - ومنها إنشاء **Function** يستدعي (**Parametrized**) **static Function**

```
#include <iostream>
using namespace std;
class clsEmployee
{
public:
    int ID;
    string Name;
    float Salary;

    clsEmployee(int ID, string Name, float Salary)
    {
        // لا يعرف Compiler هل ID تتبع ل Parameter أو Variable
        // ID = ID;

        this->ID = ID;
        this->Name = Name;
        this->Salary = Salary;
    }

    static void Func1(clsEmployee Employee)
    {
        Employee.Print();
    }
    void Func2()
    {
        // this هو Object بكامل بياناته
        Func1(*this);
    }
    void Print()
    {
        cout << ID << " " << Name << " " << Salary << endl;
        // cout << this->ID << " " << this->Name << " " << this->Salary
        << endl;
    }
};
```

```

int main()
{
    clsEmployee Employee1(101, "Ali", 5000);

    Employee1.Print();

    Employee1.Func2();

    system("pause>0");
    return 0;
}

```

Answers	Question	Quiz
True	Every object in C++ has access to its own address through an important pointer called this pointer	
True	This pointer is an implicit parameter to all member functions	
True	Therefore, inside a member function, this may be used to refer to the invoking object	
True	Friend functions do not have this pointer, because friends are not members of a class. Only member functions/member data have this pointer	
True	'this' pointer can be used to pass current object as a parameter to another method	
True	'this' pointer can be used to refer current class instance variable	

Passing Objects to Functions (ByRef/ByVal)

Objects can be treated like any data type such as int, bool, string ...etc, They can be passed to functions as parameters either **by reference** or **by value**

يمكن أن تتعامل مع **Object** مثل أي نوع من البيانات مثل int, bool, string ويمكن أيضا تمريرها ك Parameter الى Function سواء كانت

❖ **By Value (ByVal)** يتم تمرير نسخة من **Object** الى Function : يعني أي تغيير يتم على **Object** من داخل Function لن تؤثر على **Object** الأصلي – لأنها نسخة –

❖ **By Reference (ByRef)** يتم تمرير **Object** – الأصلية – الى Function : يعني أي تغيير يتم على **Object** من داخل Function سيؤثر على **Object** الأصلي – لأنها **&ByRef** –

```
#include <iostream>
using namespace std;
class clsA
{
public:
    int x;
    void Print()
    {
        cout << "The value of x=" << x << endl;
    }
};
//object sent by value, any updated will not be reflected
// on the original object
void Fun1(clsA A1)
{
    A1.x = 100;
}
//object sent by reference, any updated will be reflected
// on the original object
void Fun2(clsA &A1)
{
    A1.x = 200;
}
int main()
{
    clsA A1;
    A1.x = 50;

    cout << "\nA.x before calling function1: \n";
    A1.Print();

    //Pass by value, object will not be affected.
    Fun1(A1);
    cout << "\nA.x after calling function1 ByVal: \n";
    A1.Print();

    //Pass by value, object will be affected.
    Fun2(A1);
    cout << "\nA.x after calling function2 ByRef: \n";
    A1.Print(); }
```

Objects and Vectors : Adding Objects to Vector

```
#include <iostream>
#include<vector>
using namespace std;
class clsA
{
public:
    int x;

    //Parameterized Constructor
    clsA(int value)
    {
        x = value;
    }
    void Print()
    {
        cout << "The value of x=" << x << endl;
    }
};

int main()
{
    vector <clsA> v1;

    short NumberOfObjects = 5;

    // inserting object at the end of vector
    for (int i = 0; i < NumberOfObjects; i++)
    {
        v1.push_back(clsA(i));
    }

    // printing object content
    for (int i = 0; i < NumberOfObjects; i++)
    {
        // يوجد فيه 5 عناصر v1[i]
        v1[i].Print();
    }

    // printing object content
    for (clsA & A : v1)
    {
        A.Print();
    }

    system("pause>0");
    return 0;
}
```

Objects and Dynamic Array

قاعدة في C++ : عند استخدام New Pointer استخدامها معها delete

```
#include <iostream>
using namespace std;
class clsA
{
public:
    // dummy constructor
    clsA() {}

    //Parameterized Constructor
    clsA(int value)
    {
        x = value;
    }

    int x;

    void Print()
    {
        cout <<"The value of x="<< x <<endl;
    }
};

int main()
{
    short NumberOfObjects = 5;

    // allocating dynamic array
    // of Size NumberOfObjects using new keyword
    clsA * arrA = new clsA[NumberOfObjects];

    // calling constructor
    // for each index of array
    for (int i = 0; i < NumberOfObjects; i++)
    {
        arrA[i] =clsA(i);
    }

    // printing contents of array
    for (int i = 0; i < NumberOfObjects; i++)
    {
        arrA[i].Print();
    }

    // delete استخدام Pointer لا يتم حذفها من الذاكرة إلا بشكل يدوي باستخدام delete
    delete []arrA;

    system("pause>0");
    return 0;
}
```

Objects with Parameterized Constructor and Array

```
#include <iostream>
using namespace std;
class clsA
{
public:
    //Parameterized Constructor
    clsA(int value)
    {
        x = value;
    }

    int x;

    void Print()
    {
        cout <<"The value of x="<< x <<endl;
    }
};

int main()
{
    // Initializing 3 array Objects with function calls of
    // parameterized constructor as elements of that array
    clsA obj[] = { clsA(10), clsA(20), clsA(30) };

    // using print method for each of three elements.
    for (int i = 0; i < 3; i++)
    {
        obj[i].Print();
    }

    system("pause>0");
    return 0;
}
```


#Lesson 38 – 39 :String Library Project (Requirements & Solution)

```
class clsString
```

<https://cdn.fs.teachablecdn.com/hlZrA1gVShuz8qvpN8L8>

```
int main()
```

<https://cdn.fs.teachablecdn.com/fG9l1hEMSuuPWqEfGnR2>

#Lesson 40 – 41 : Date Library Project (Requirements & Solution)

```
class clsDate
```

<https://cdn.fs.teachablecdn.com/HePhw0MSSxu8Xz4DJXtd>

```
int main()
```

<https://cdn.fs.teachablecdn.com/WcEh7OeQRrKZ9DlIBiCB>

The following is source code for Period class and the way to use it

```
class clsPeriod
```

<https://cdn.fs.teachablecdn.com/OtWIAMSGSwGsk3ZeMcX0>

```
class clsDate
```

<https://cdn.fs.teachablecdn.com/HePhw0MSSxu8Xz4DJXtd>

```
int main()
```

<https://cdn.fs.teachablecdn.com/LY58VdzTo2saVhBU6FI9>

#Lesson 42 : What is the difference between Class and Structure ?

ما هو الفرق بين **Class** و **Structure** ؟

لغة **C++** امتداد الى **C** والفرق الذي بينهما هو أن لغة **C++** تدعم **OOP**

أما لغة **C** لا تدعم **OOP** ولكن يوجد فيها **Structure** ووظيفتها جمع Variables في مجموعة واحدة ولكن لا تستطيع في **Structure** في لغة **C** أن تضع داخلها Function & Procedure (**Method**)

أما في لغة **C++** فقد تم بناء **Class** على **Structure** الموجود في **C** ، وبما أنهم قد طوروا **Structure** في **C** بجعله **Class** فقد أضافوا كل المميزات التي في **Class** الى **Structure** الموجود في **C++**

ولكن يوجد بينهما أي **Class** و **Structure** في **C++** اختلافات بسيطة

(متى تستخدم **Class** ومتى تستخدم **Structure** ؟)

استخدم **Structure** للبيانات ذات المساحة الصغيرة فقط (للمتغيرات Variables فقط)
و **Structure** لا يوجد في كل لغات البرمجة

واستخدم **Class** للبيانات ذات المساحة الكبيرة (أو **Method** ولو كان واحد)

أي شيء تريده من **OOP** استخدم **Class**


Structure	Class
تنشأ ب : Struct	تنشأ ب : Class
تعرف Members التي بداخلها بشكل افتراضي Public :	تعرف Members التي بداخلها بشكل افتراضي Private :
الهدف الرئيسي هو : تجميع Data بداخل مجموعة	الهدف الرئيسي هو : <u>Abstraction</u> و <u>Inheritance</u>
يتم تخزينه في Stack	يتم تخزينه في Heap
تستخدم فيها <u>Parameterized Constructor</u> فقط	تستخدم فيها جميع أنواع <u>Constructor</u>


Struct

VS

Class

Keyword: struct	Keyword: class
Default: Public	Default: Private
Purpose: Generally for Grouping Data	Data abstraction and more inheritance
Value Type (Stack)	Reference Type (Heap)
Only Parametarized Constructors.	All types of constructors.


PROGRAMMING
ADVICE
LEARN AND GROW WITH US



Mohammed Abu-Hadoud
MBA, PMP®, Scrum®, PRINCE2®, Agile, C#, TFS, ASP.NET, MVC5
25+ years of experience

جدول للمصطلحات البرمجية + المبادئ OOP

مصطلحات برمجية	+ مفاهيم أو مبادئ OOP	
<u>OOP</u>	Object Oriented Programming	البرمجة الشيئية أو الكنائية
<u>Method</u>	هي Function & Procedure الموجودة داخل Class	
<u>Members</u>	هي Variable & Method	Object فيه محتويات وتسمى Members
<u>class</u>	فئة يندرج تحته كل Members التي لها علاقة بهذه الفئة	عبارة عن Datatype
<u>Object / Instance</u>	هو اسم Variable ل Class	ClassName ObjectName
<u>Access Specifiers (Modifiers)</u>	هو ثلاثة أنواع : خاص ، محمي ، عام	Private , Protected , Public
<u>Modifier Private</u>	تستطيع الوصول الى Members من داخل Class فقط	
<u>Modifier Protected</u>	يستطيع كل Classes التي ترث Inheritance هذه Class رؤية هذه Members + من بداخل Class	
<u>Modifier Public</u>	أي أحد يستطيع الوصول إلى Members من أي مكان في البرنامج	
<u>Property Set</u>	للتعديل على Variable من داخل Class باستخدام Function	
<u>Property Get</u>	للحصول على Variable من داخل Class باستخدام Function	
<u>Encapsulation</u>	هي Class التي تجمع كل Methods ذات العلاقة تحت سقف واحد ، ولها علاقة أيضا في إخفاء البيانات التي هي Property	
<u>Abstraction</u>	إخفاء جميع الأشياء غير الأساسية أو غير المهمة قدر الإمكان في Method عن طريق Object للمستخدمين	
<u>Constructor</u>	هو Function موجود في Class يتم استدعاؤه كلما عرفت Object من نوع Class ، يتم إنشاؤه إما بشكل افتراضي عن طريق Compiler أو أنت الذي تنشئه	

هو آخر Function ينادى في Class لتدمير Object أو أي شيء آخر ، وطريقة كتابته { تنفيذ آخر الأوامر } ~ClassName()	<u>Destructor</u>
هو Variable مشترك لجميع Objects، وأي Object يعدل عليه يتم تعديله على جميع Objects الأخرى (لابد من تعريف Variable خارج Class ك Global Variable لا يكون بداخل Function)	<u>Static Variable</u>
تستطيع مناداته من غير Object (clsName::FunctionName)	<u>Static Function</u>
إنشاء Class جديدة ترث كل Members (Public & Protected) الخاصة ب Class القديمة	<u>Inheritance</u>
Base Class هي التي يتم توريث خصائصها Classes أخرى	<u>Super Class / Base Class</u>
هي Derived Class التي ترث الخصائص من Base Class	<u>Sub Class / Derived Class</u>
هو : إلغاء أو تجاوز Function في Base Class الى Function الخاص ب Derived Class (يكون بنفس الاسم)	<u>Function Overriding</u>
هو التحويل من Derived Class الى Base Class عن طريق Pointers مثال : clsBase * ObjectBase = & ObjectDerived	<u>Up Casting</u>
يتم إنشاء مكان في الذاكرة اسمه Virtual table يخزن فيه كل Address الخاصة ب Overriding Method لكيلا يحدث خطأ أثناء الاستدعاء	<u>Virtual Functions</u>
تم ربط Address قبل تشغيل البرنامج ب Member التابع ل Class الخاص Object	<u>(Static / Early) Binding</u>
يتم ربط Address وقت تشغيل البرنامج مثاله : Virtual Functions	<u>(Dynamic / Late) Binding</u>
يسمح Polymorphism للكود بالتعامل في Object مع أنواع مختلفة دون الحاجة الى كتابة كود مختلف لكل نوع	<u>Polymorphism</u>
إجبار من يرث منها على تنفيذ كل الشروط التي هي Pure Virtual Function بإنشاء Overriding في Derived (بمجرد إنشاء Pure Virtual : لا يمكنك إنشاء Object منها)	<u>أو Interfaces</u> <u>أو Abstract Classes</u> <u>Contract</u>

يستطيع clsB الوصول الى كل Members التي بداخل clsA ولو كانت Private & Protected من داخل clsB فقط (وكذلك Friend Function)	<u>Friend Classes</u>
هي Class التي تحتوي Class الداخلية	<u>Enclosing Class</u>
هي Class التي بداخل Enclosing Class	<u>Inner Class</u>
يستخدم للوصول الى كل Data Members في داخل Class : this->Members	<u>'this' Pointer</u>

قد يكون هنالك بعض الأخطاء سواء في الكتابة – خاصة في اللغة الإنجليزية – أو قد يكون هناك خطأ في المعلومة فارجوا التصحيح

هذا وصلى الله على نبينا محمد وعلى اله وصحبه أجمعين ، وآخر دعوانا أن الحمد لله رب العالمين