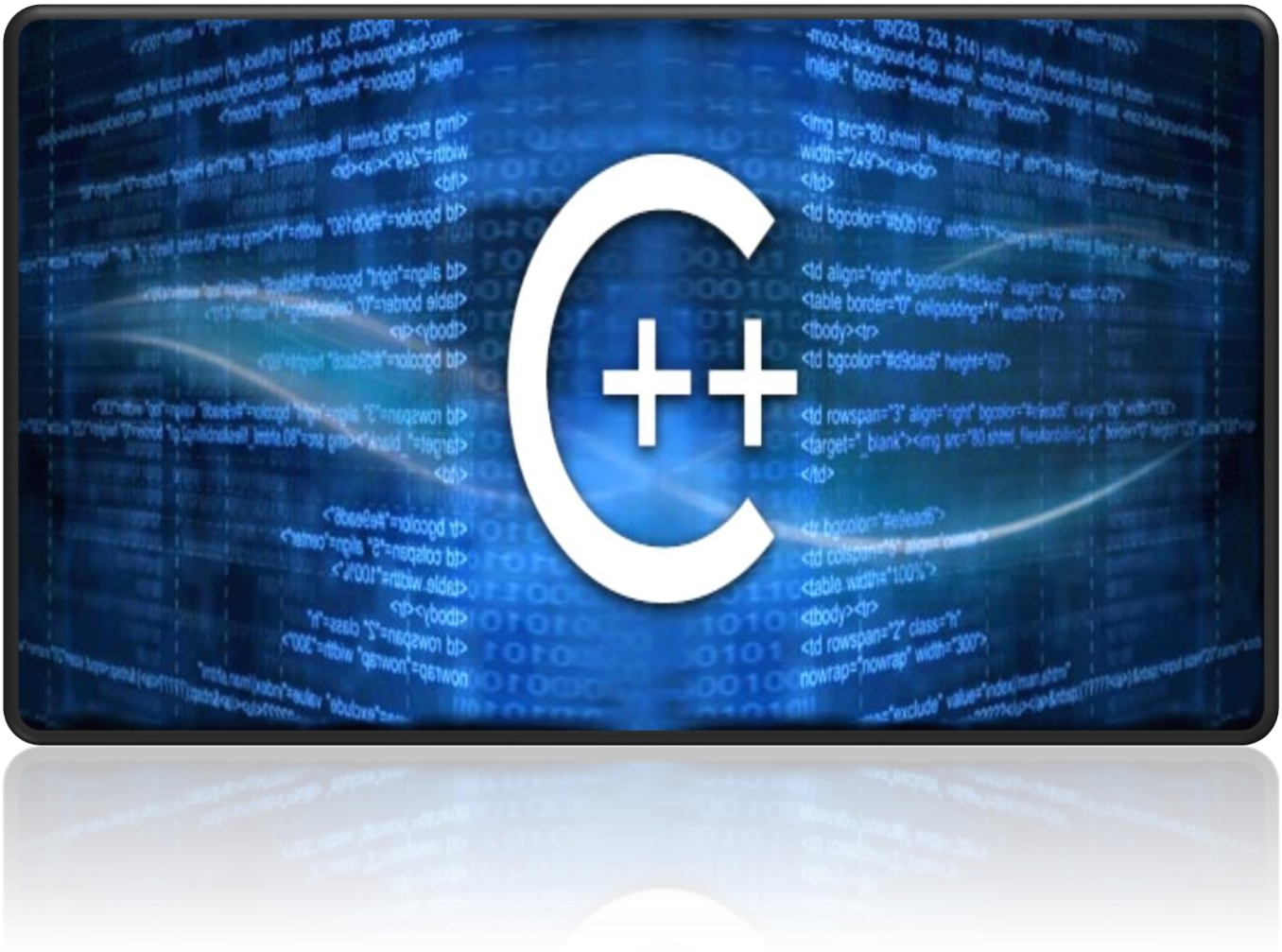
Object-Oriented Programming

in C ++

Cookbook



Content

About writer : -

Name : Mahmoud said

Nationality: Egypt

Job : instructor of computer science

Full stack developer

logic design and algorithms

How To Content :

https://www.facebook.com/mahmoud.said.NST

My channel :

Name : new system technology

URL : https://www.youtube.com/channel/UCSu-YBmsmjSlPTKtlKx_img


NST
NEW SYSTEM
TECHNOLOGY
LINUX

Suggestions:

ارحب بكل المقترحات او النقد الهادف للكتاب فيمكنكم التعليق و ارسال رسالة لي علي صفحتي الشخصية علي facebook

لا اقبل الاتي في الرسائل :

الملفات او أي شيء يمكن ان أقوم بتحميله

استخدام الروابط المختصرة أياً كان الموقع موثوق به

Introduction

هل سألت نفسك مرة لماذا لا اقدر ان افهم مكتبة برمجية معينة ؟؟

او عندما تبحث عن كود لا تعرف كيف تقوم بكتابته ثم تقوم بفتح الكثير من المراجع والقراءة بها ولكن لا تفهم لماذا هذه الاكواد مكتوبة بهذه الهيئة وهذا الشكل ؟

فهذا يدل علي انك تعرف البرمجة ولكنك لا تعرف الكثير عن الشكل البرمجي الخاص باللغة التي تقوم باستعمالها .

اذا هناك سؤال بديهي ... ولماذا اتعلم الشكل العام لكتابة الاكواد فأي كود ينفذ يفي بالغرض؟؟

نعم انت يمكنك ان تقوم بالكتابة كيفما تشاء ولكن ليست الفائدة الوحيدة الان هي مجرد التحكم في الالة فقط او عمل بعض البرامج. لان لغات البرمجة اصبحت اللغات التي يفهمها المبرمجين في العالم فاذا حاولت الان ان تقوم بكتابة اكواد علي هيئة غير متعارف عليها ثم أعطيت هذا الكود لمبرمج اخر سيتعب كثيرا حتي يفهم ما الذي كنت تريد تنفيذه من خلال هذا الكود .

لذلك يجب ان تتعلم كيف تستطيع تركيب الاكواد وكيف تقوم باستخدام الشكل العام حتي تقوم بفهم هذه الاكواد مرة اخري اذا تركتها لفترة او اذا اعطيتها لمبرمج اخر

وحتي تقوم أيضا بفهم المكاتب البرمجية حتي تستخدمها .

تسال الان عن كيف اتعلم هذه الصيغ والقواعد التي تقوم عليها أسس أسلوب البرمجة الان **. اذا فعليك ان تقوم بتعلم أسلوب البرمجة الكائنية وهذا ما سنتطرق اليه في هذا الكتاب .**

CHAPTER 1

Why learn Object-Oriented

1 – لماذا البرمجة الكائنية

2 – الفرق بين البرمجة الاجرائية والبرمجة الموجهه

3 – من منظور الشركات و فريق العمل

4 – مميزات البرمجة الكائنية

5 – ملخص الكلام

لماذا البرمجة الكائنية :-

السؤال الذي حير الكثير من المبرمجين حديثي الدخول لعالم البرمجة الكائنية او البرمجة بشكل عام لما علي استخدام الأساليب المختلفة في البرمجة ؟

نستخدم هذه الأساليب لنجعل الاكواد اكثر تنظيم واكثر فهم واختصار في الكتابة وكذلك هي لغة تفاهم بين فريق العمل فلا يعمل أي فريق علي مشروع الا ويستخدمون هذا الأسلوب . اذا هل انت تحتاج اليه وانت تعرف ان البرامج التي تقوم بعملها لا تتجاوز 100 اسطر فقط ؟؟

الإجابة هي نعم . فانت اذا حاولت ان تقوم بالتطوير في البرنامج لكي يقوم بالعمل بطريقة اسرع او ان يتواجد به اكثر من حل لأي مشكلة تواجهه التي صمم من اجل حلها فانت اذا كنت لا تستخدم البرمجة الكائنية فستحتاج الي إعادة كتابة البرنامج حتي يتماشى مع الرؤية الجديدة او الحل فلماذا هذا العناء وانت يمكنك ان تقوم ببناء برنامج مميز تقوم بتطوير كل جزئية به منفصلة عن اختها هكذا تقوم بعمل برنامج اكثر ثبات ودقة .

الفرق بين البرمجة الإجرائية و الموجهة**:**

سوف اريكم نوعين من الاكواد كلاهما لهما نفس التأثير ولكن كتبا بطريقة مختلفة لكي نري الفرق .

النوع الأول البرمجة الإجرائية :-

```
#include<iostream>

using namespace std;

int main(int argv,char **argc)
{
	int width;
	int length;
	int area;
	cout << "please enter (width):" << endl;
	cin >> width;
	cout << "please enter (length):" << endl;
	cin >> length;
	area = length*width;
	cout << " your area is : " << area << endl;

	return 0;
}
```

هذا الكود بسيط قد كلفني بعض الخطوات والاسطر البسيطة داخل الدالة الرئيسية

ولكن تخيل معي اذا احتجنا ان نقوم بعمل هذه العملية 10 مرات سوف نقوم بنسخ ولصق هذه الاسطر 10 مرات هذا هو المرهق في هذا النوع من البرمجة القديمة .

اذا ما رايكم اذا اختزلنا هذه الاكواد داخل كبسولة كل مرة نستدعي هذه الكبسولة حتي ينفذ الكود بمعني اننا سوف نقوم بكتابته مرة واحدة فقط وكل مرة نقوم باستدعائه سوف نكتب فقط امر او اثنين في الاستدعاء هكذا وفرنا في جهد الكتابة والتعديل واصبح مفهوما اكثر .

الطريقة الثانية البرمجة الموجهة: -

```
#include<iostream>

using namespace std;

using namespace std;
class Rect{
	public:
		int width;
		int length;
		int area;
	    void get_area()
		{
			cout << "please enter (width):" << endl;
			cin >> width;
			cout << "please enter (length):" << endl;
			cin >> length;
```

```
                area = width*length;
                cout << " your area is : " <<  area << endl;
            }
};
int main()
{
    // code call only
      Rect r1; r1.get_area();
   return 0;
}
```

من منظور الشركات و فريق العمل :

من اكثر الأسئلة الشائعة عند تقدمك لوظيفة مبرمج داخل أي شركة تتحدث معك عن مفهوم البرمجة الكائنية وعن حل المشكلات ولكن دعونا من جزئية حل المشكلات فلدينا كتاب كامل يتحدث عن هذه الفقرة ولكن الان الشركات تستخدم احدث أنظمة التشغيل واحدث الأدوات البرمجية وبيئات التطوير حتي تسهل عليك البرمجة ولكن يجب ان تضمن انها حين تجعلك من موظفيها انك مؤهلا لاستخدام هذه التقنيات وهي تعتمد اعتماد كامل علي أسلوب البرمجة الموجه.

اما بالنسبة للفريق فكما ذكرنا من قبل فهي لغة التفاهم الخاصة بالمبرمجين فانت لن تقوم ببرمجة برنامج كامل بنفسك فكيف نقوم بجعل كل شخص من الفريق يعمل علي الجزئية الخاصة به وهو عن طريق البرمجة الموجه فيمكن ان نقوم بجمع وتركيب جميع الاكواد التي انشاها الفريق ومن ثم تجربتها وتحسين أدائها الخ .. كذلك قد وفرنا الكثير من الوقت والجهد واصبح البرنامج سهل القراءة والفهم ويمكن التعديل عليه بسهولة .

مميزات البرمجة الكائنية :

1. التغليف (Encapsulation).
2. إخفاء البيانات (Data Hiding).
3. الميراث (Inheritance).
4. تعددالأشكال (Polymorphism).

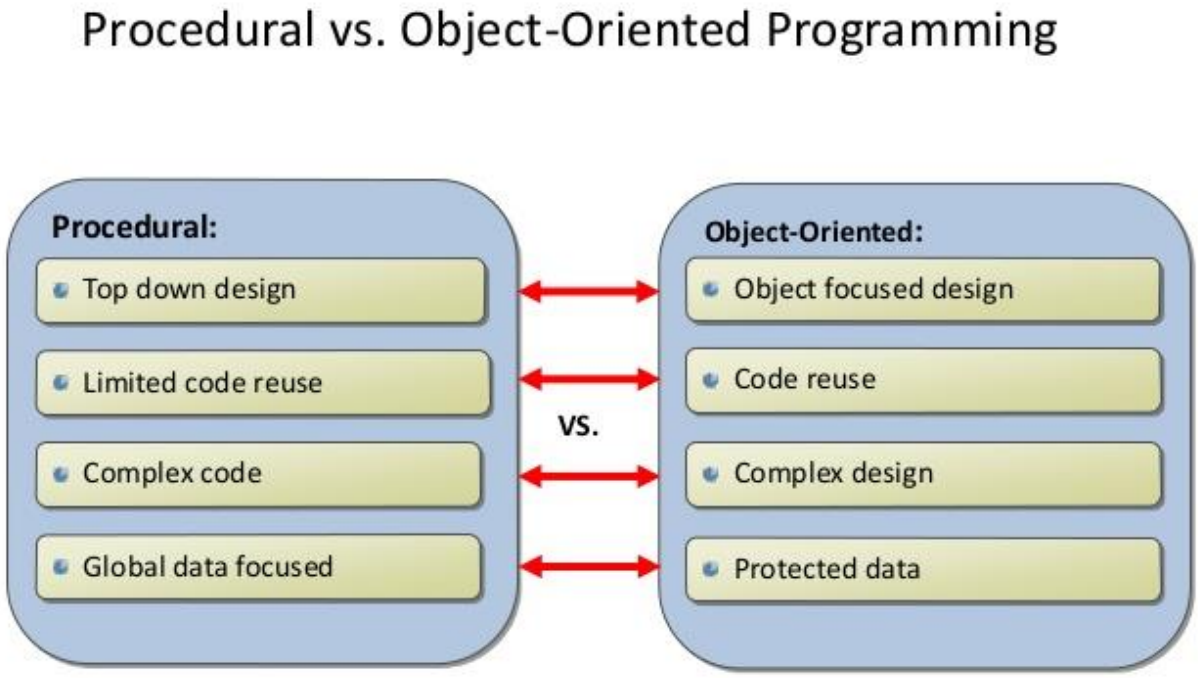
ملخص الكلام :

1 – يوجد فرق بين البرمجة الإجرائية والبرمجة الكائنية

2- البرمجة الكائنية الموجهة مطلوبة من الشركات وفريق العمل

3 – اكتب هيكل الكود مرة واحد ثم قم باستدعائه بدلا من كتابة الهيكلة كل مرة

وأخيرا هذه الصورة توضح الكثير



CHAPTER 2

definitions about Object-Oriented

1 – ما هو الـ Class

2 – ما هو الكائن (Object)

3 – ما هي الـ Methods

4 – ما هو الـ Encapsulation

5 – ما هي الوراثة Inheritance

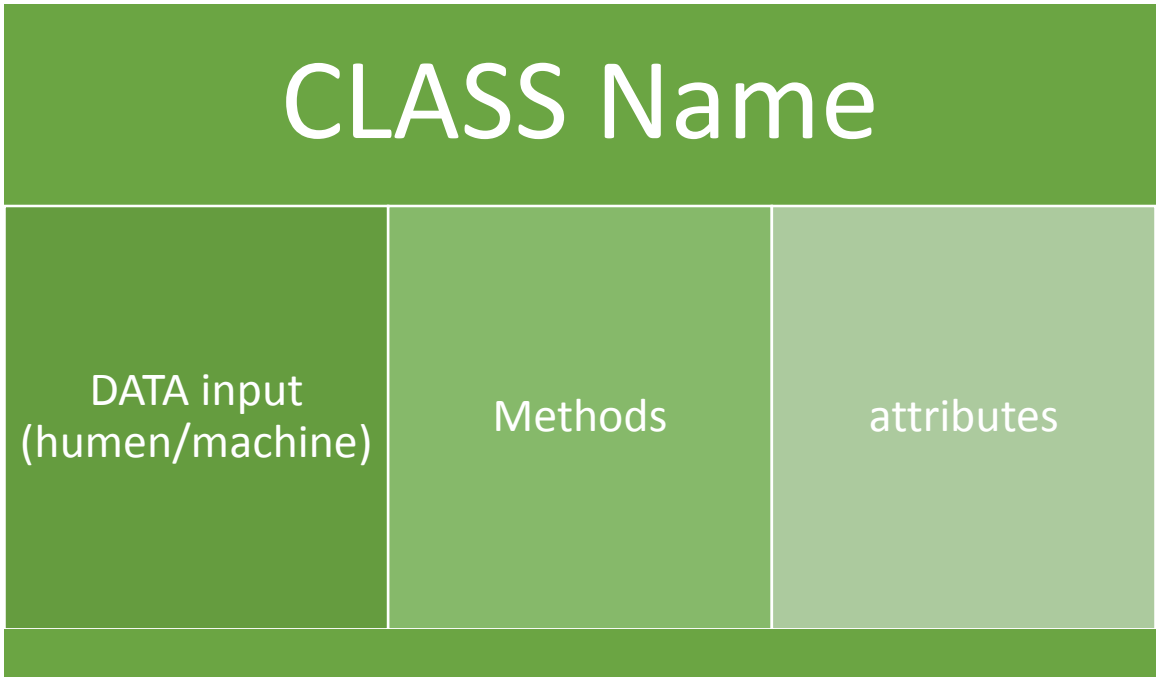
6 – ما هو تعدد الاشكال Polymorphism

ما هو الـ CLASS

هو عبارة عن قالب تقوم بتخزين به مجموعة من الأوامر والتي تنقسم الي قسمين خصائص و سلوكيات .

لكل كلاس اسم حتي يسهل الوصول اليه او معرفة وظيفته من قبل ان تستخدمه.

لكل كلاس هيكل يضم البيانات والوظائف التي تتم علي هذه البيانات .



ولماذا احتاج الي كلاس ؟

حتي أقوم بتجميع الأوامر البرمجية وتصنفيها حيث يمكن ان تجعل الاكواد الخاصة بالشبكات في كلاس و تسميه باسم محدد وكذلك الاكواد الخاصة بالجرافيك والاكواد الخاصة باتصال قواعد البيانات .

ما هو الكائن(Object)

الكائن هو نسخة او صورة من الكلاس تصنع داخل الذاكرة بحيث ان الكائن يوجد لديه نفس الخصائص والسلوكيات بداخل الكلاس .

لماذا أقوم بعمل كائن ؟

حتي أقوم باستدعاء جميع الخصائص والسلوكيات التي قمت بتعريفها في الكلاس

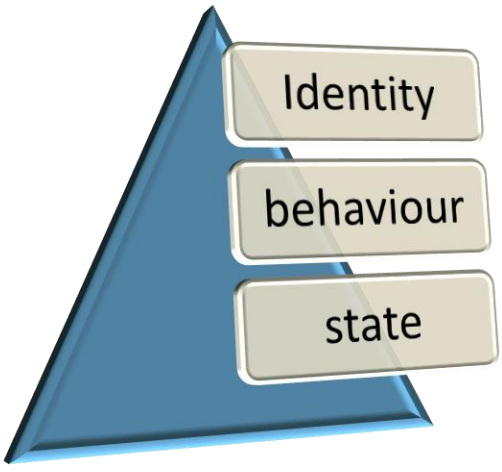
مثال : اذا كان لديك ورقة تريد ان تضع بها بياناتك وأرسالها الي بعض الشركات حتي يتعرفوا علي ما هي صفاتك والوظائف التي تقدر ان تقوم بها ولكن مع مراعات ان تقوم بتغير أولويات الوظائف علي حسب كل شركة واحتياجاتها فيجب عليك ان تقوم بتصوير هذه الورقة او طباعة اكثر من نسخة ولكنك لن تقوم بكتابتها مرة اخري بل تقوم فقط بعمل منها صورة .. اذا هذا هو الكائن .

قوانين كتابة الكائن :-

لكل كائن اسم خاص به لا يمكن إعادة استخدام الاسم لكائن اخر

الصفات – له معلومات مخزنة

السلوكيات – لكل كائن وظائف يقوم علي تنفذها



ما هي الـ Methods

هي مجموعة من الوظائف تكتب داخل الكلاس تضم بعض البيانات الخاصة بها وتستخدم بعض البيانات العامة التابعة للكلاس يمكن ان تقوم بإرسال اليها بعض البيانات لكي تقوم بعمل معالجة و تقوم بإرجاع قيمة او نتيجة هذه العملية.

خصائصها :-

1 – لها اسم ولا يمكن ان تستخدم هذا الاسم مرة اخري الا باستخدام بعض التكنيك المختلفة (overloading, overriding)

2 – يجب تحديد نوع البيانات التي تعمل عليها معالجة

3 – لا يمكن استدعائها الا من خلال عمل كائن من كلاس او استخدام اسم الكلاس نفسه .

ما هو الـ Encapsulation

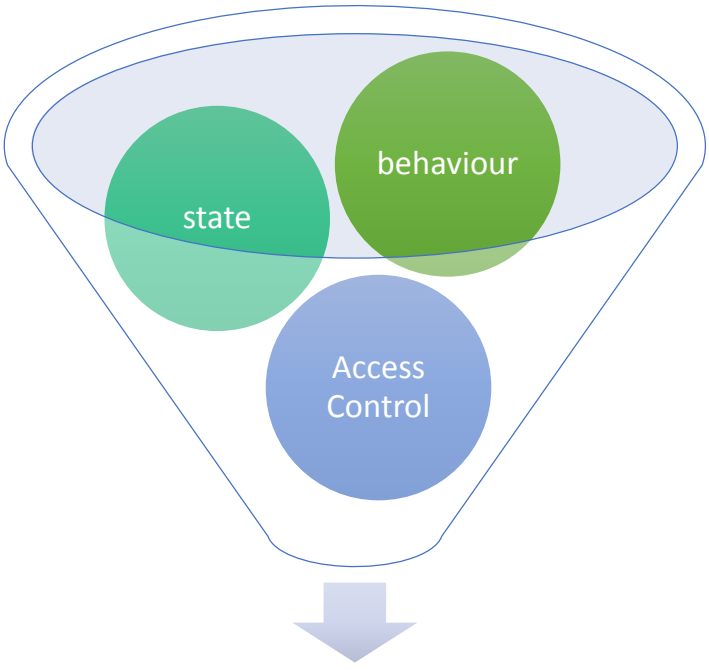
هو عبارة عن مجموعة تتشابه في الخصائص والسلوكيات مخزنة في Module

اذا ما فائدة هذا التكنيك ؟

هو تجميع جميع الصفات والسلوكيات داخل كلاس.

عمل تحكم علي ظهور المعلومات والسلوكيات والتحكم في الوصل اليها .

فهو يقوم علي عمل إخفاء لبعض المعلومات او التحكم في طريقة معالجتها للبينات المدخلة حتي لا تحدث أخطاء بين المبرمجين الذين يقوموا بتطوير كلاس معين .



Encapsulation

ما هي الوراثة (Inheritance)

هي السماح بصنع كلاس جديد بواسطة كلاس قديم موجود بالفعل .

تقوم بوراثة كل شيء بداخل الكلاس القديم من خصائص وسلوكيات بمراعاة إمكانية الوصول الي بعض الخصائص والسلوكيات التي يحتويها الكلاس القديم .

ولكن يجب ان يكون داخل الكلاس الجديد ما يميزه بحيث لا يقوم باستعمال نفس الاسم ويكون بداخله بعض الخصائص والسلوكيات الخاصة به .

حينما يكون لديك كلاس بهه بعض خصائص وسلوكيات التي يمكن ان تكرر في الاكواد فيمكنك ان تقوم باستغلال هذا الكلاس بحيث تقوم بأخذ كل ما بداخله من غير ان تكرر كتابته مرة اخري وان تضيف ما تريده في الكلاس الجديد مثال :

لدينا كلاس لأنواع الحيوانات اسمه Animal ويوجد بداخله الصفات الاتية :

الاسم و النوع و الصوت . ويوجد لديه هذه السلوكيات :

(do_walk, do_stop , do_sit , do_sound)

كل هذه السلوكيات والخصائص توجد في جميع الحيوانات فلا يوجد حيوان بلا صوت او اسم . المشكلة هنا عندما تقوم بتخصيص نوع محدد من الحيوانات فاذا تحتاج ان تقوم بعمل كلاس يعبر عن الكلاب مثلا فسوف تقوم بنسخ ولصق كل هذه الصفات والسلوكيات وإعادة كتابتها مرة اخري . فعندما تشعر بانه يوجد تكرار داخل الكود الخاص بك فاعلم انك لست علي الطريق الصحيح .فما الحل ؟؟

هو ان تقوم بعمل كلاس جديد وتقوم بوراثة الصفات التي توجد في الكلاس القديم الذي قد انشاته وهكذا قد نجحنا في حل مشكلة تكرار الكود واصبح لديك بعض التنظيم واختلاف مستوي كفاءة البرنامج .

ما هو الـ Polymorphism

هو تكنيك برمجي يسمح ان تقوم بعمل أوجه كثيرة لنفس الكائن او الاجراء البرمجي .
والتسمية العلمية الخاصة به هو تعدد الاشكال.

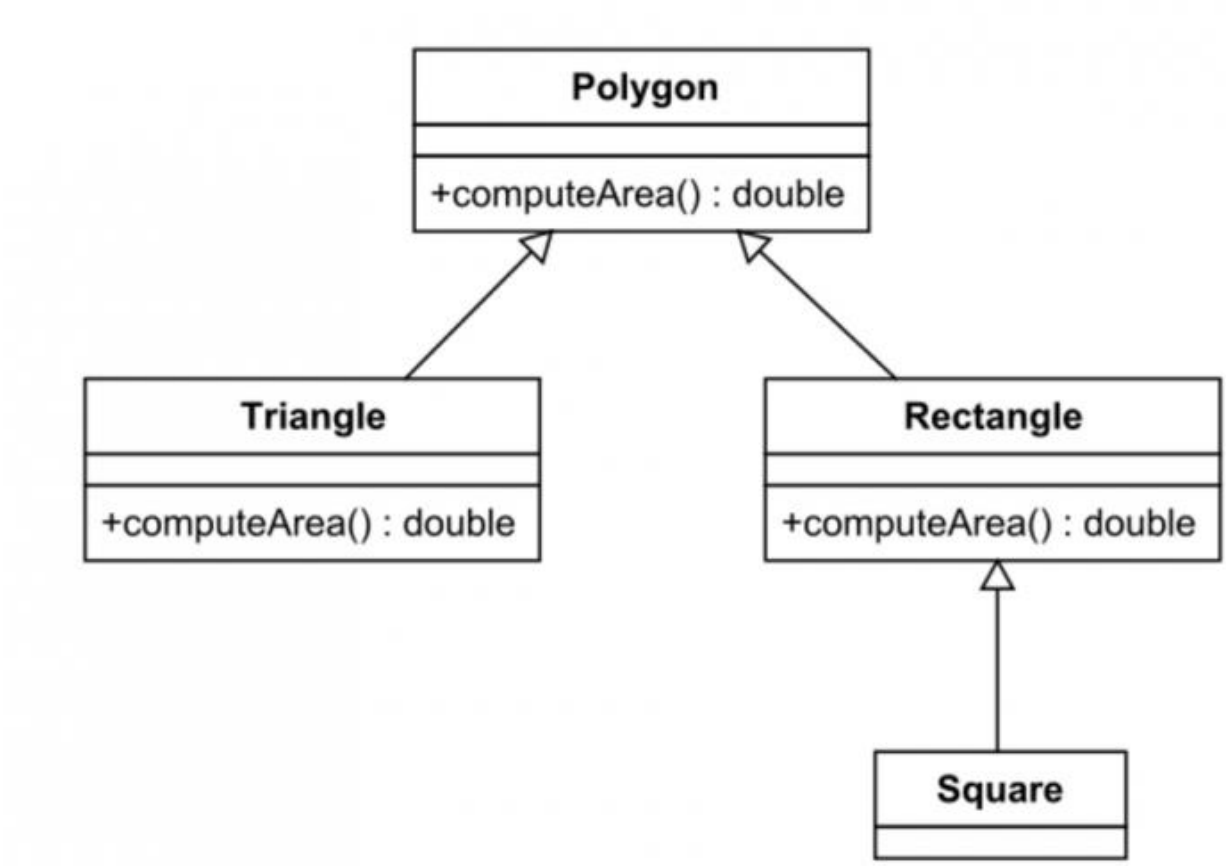
POLYMORPHISM MEANS THAT SOME CODE OR OPERATIONS OR OBJECTS BEHAVE DIFFERENTLY IN DIFFERENT CONTEXTS.

وهو يتمثل في لغة السي ++ في ثلاث حالات هما :

- Overloading
- Overwriting/Overriding
- Instance from child-Class pointer to parent-Class

1 – تكرار الـ Methodsبنفس الاسم ولكن تغير في عدد الـ Parameters
او نوع الرجوع بالقيم .
2 – استخدام Virtual methods
3 – استخدام New keyword

كما موضح في هذا الشكل كيف يقوم الكائن بالإشارة الي كلاس اخر عندما يكون الكلاس الخاص بالكائن قد ورث من الكلاس الذي يشير عليه الكائن وهذا ما يجعل له الصلاحية بان يقوم باستخدام بعض خصائص الكلاس .



او كما موضح في هذه الاكواد :

```
Employee *emplP;
emplP = &empl;  // make point to an Employee
cout << "Pay: " << emplP->pay(40.0);  // call Employee::pay()
emplP = &mgr;  // make point to a Manager
cout << "Pay: " << emplP->pay(40.0);  // please--Manager::pay()?
```

CHAPTER 3

The difference between structure & Classes

- كيف تقوم بعمل structure
- كيف تقوم بعمل Class
- ما الاختلاف بينهم وايهم افضل

كيف تقوم بعمل Structure

اذا قد وصلنا أخيرا الي قسم الاكواد والذي يفضله الكثير من المبرمجين عن التعاريف والاشياء النظرية ولكن دائما ما أقول انه يجب ان نقوم بتعلم البرمجة وليس تعلم الاكواد (لا تتعلم الكود ولكن تعلم كيف تصنعه) هذا هو مبدأنا من البداية وحتي تصبح مبرمج قوي في مجالك يجب ان تقوم بتعلم النظريات أولا لتكون قادر علي تنفيذها .

ما هو الهيكل :

هو أحد أنواع هياكل البيانات , يمكن أن يضم أنواعا متغيرة من البيانات .

هو انك تقوم بعمل نوع جديد من أنواع البيانات ولكنه يضم بعض المتغيرات والتي تضم أنواع بيانات اخري مجتمعين داخل نوع جديد انت قمت بصنعه من خلال تنفيذك لكود الهيكل.

كيف تكتب الهياكل :

```
struct NameOfStruct{
	// Code
	// هنا تقوم بكتابة المتغيرات
	int testNum;
};
```

كيف نقوم بتعريف الهيكل داخل الدالة الرئيسية :

```
int main()
{
	NameOfStruct instance;
	instance.testNum = 5;
	cout << "print value : " << instance.testNum << endl;
	return 0;
}
```

كيف تقوم بعمل Class

```
class nameOfClass{
      public:
      int width;
      int legnth;
      int area = width * legnth;

};
```

ملحوظة :

كلمة Publicهي نوع من أنواع Access modifier

والذي سوف نتعرف عليه لاحقا .

كيف أستخدم الكلاس داخل الدالة الرئيسية :

```
int main(){
      nameOfClass object1;
      object1.width = 5;
      object1.legnth =4;
      int calc_area = object1.area;
      cout << calc_area << endl;
return 0;
}
```

ما الفرق بين الهيكل والكلاس :

يوجد مميزات كثيرة في الكلاس لم تكن موجودة بالهياكل ولكن الأهم في انهم مجتمعين علي انك يمكنك ان تقوم بصنع كائن او نوع جديد من المتغيرات .

ولكي توضح الصورة اكثر انظر في هذا الجدول الذي يقارن الكلاس بالهيكل :-

Structures	classes
جميع المتغيرات والدوال بداخله يمكن لأي احد ان يقوم باستخدامها (لا يوجد مفهوم الكبسولة)	يمكن التحكم في الوصول الي بعض المغيرات والدوال ويمكن جعلها خاصة او محمية
لا يمكن الوراثة منه	يستخدم الوراثة و الوراثة المزدوجة
لا يمكن جعل الكائن ان يكون لديه عددت حالات مختلفة polymorphism	يتبع أسلوب تعدد الأوجه والحالات
لا يدعم Interface	يدعم **Interface class and abstraction class**

CHAPTER 4

using Classes

1 – ما هو UML

2 – ما هو class diagram

3 – ما هي العلاقات وكيفية تصميمها

4 - انشاء كلاس لتكوين لعبة

ما هو UML:

لغة التصميم الموحدة تقدم صيغة لوصف العناصر الرئيسية للنظم البرمجية.

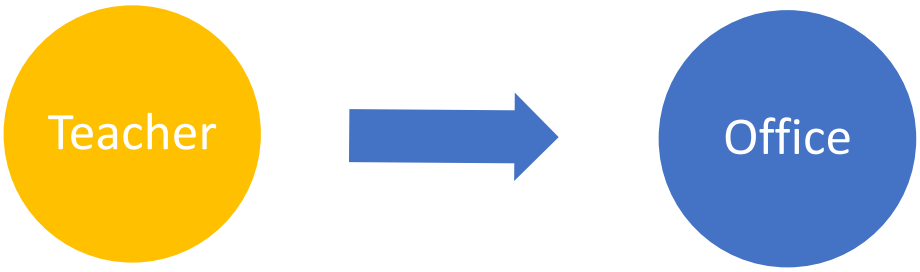
تستخدم هذه اللغة لعمل رسوم تخطيطية لوصف برامج الكمبيوتر من حيث العناصر المكونة لها أو خط سير العمليات الذي يقوم به البرنامج.

أنواع العلاقات المستخدمة :

النوع الأول : One-to-One

في هذا النوع من أنواع العلاقات هو ان تقوم بصنع علاقة كائن واحد مع كائن اخر مثال علي هذا النوع .

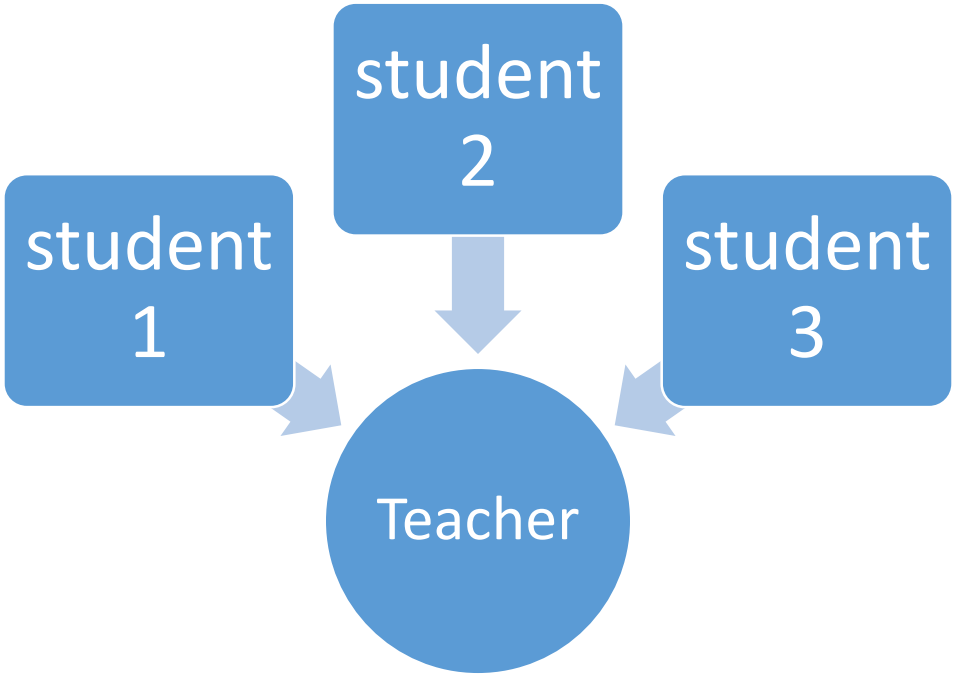
علاقة المدرس بالمكتب الخاص به فلكل مدرس في المدرسة او الجامعة مكتب خاص به واحد فقط .



النوع الثاني : One-to-Many

في هذا النوع من العلاقات هو ان تقوم بصنع علاقة بين كائن ومجموعة من الكائنات التي تتشارك في بعض الخصائص .

مثال المدرس وعلاقته بالطلبة فلكل مدرس لديه عدد من الطلاب يقوم بتعليمهم
فالمدرس كائن واحد والطلبة مجموعة من الكائنات يتشاركون في صفات الطالب .



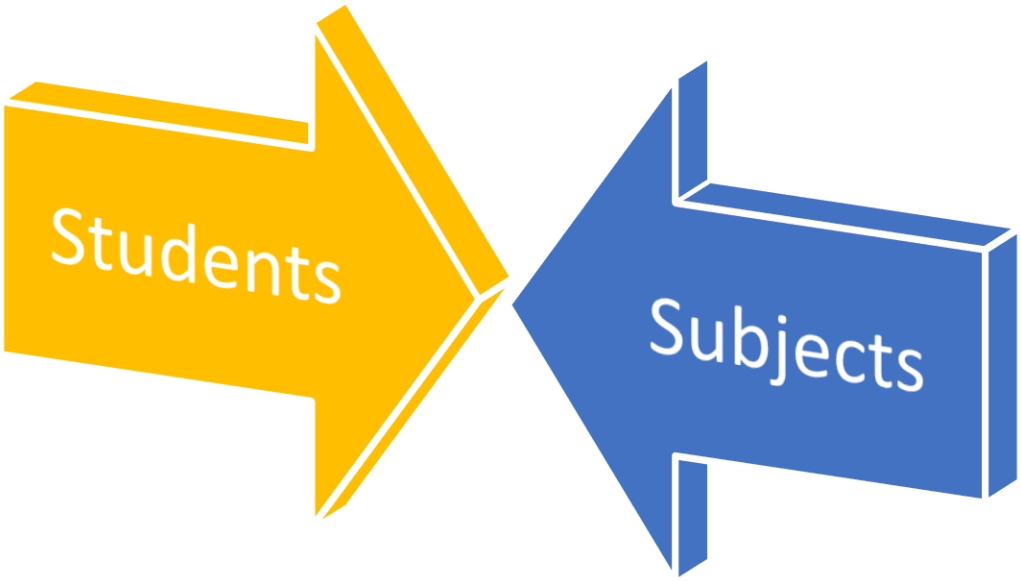
النوع الثالث : Many-to-Many

هذا النوع من العلاقات يكون بين مجموعة من الكائنات ومجموعة اخري ويرمز اليه

M →N OR (* → *)

يمكن ان يكون عدد الكائنات الاولي ليست مساوية لعدد الكائنات الأخرى مثال :

الطلاب و علاقتهم بكمية المواد الخاص بهم .



ما هو class diagram

يستخدم على نطاق واسع لوصف أنواع الكائنات "Class Diagram" مخطط الفئة الموجودة في النظام و علاقاتها ببعضها "objects"

وتتألف الفئات من ثلاثة أشياء :

"operations" ، والعمليات"attributes" والصفات "name"

"class" هذا مثال على الفئة .

ما هي العلاقات وكيفية تصميمها :

- **Aggregation : هو العلاقة بين الكلاس الوالد والكلاس الابن بحيث يقدر ان يكون كلاس الابن مستقل عن كلاس الاب اذا قد تم حذف كلاس الوالد فلا يؤثر علي وجود كلاس الابن .**
- **Composition: هو علاقة الكلاس الابن بكلاس الوالد بحيث اذا تم حذف كلاس الاب لن يكون بمقدرة كلاس الابن بالعمل مثال : البيت اذا لم يكن موجود فلن يكن للغرف وجود .**
- **Dependency : الاعتمادية هي تكون اضعف بكثير من العلاقة فهي تقوم فقط علي استقبال المعاملات او نوع البيانات الراجع .**

انشاء كلاس لتكوين لعبة :

الان سوف نقوم بعمل تصميم لهيكل لعبة ولكن عن طريق تصميم البرمجة الكائنية .

المتطلبات :-

1 – كلاس اللعبة

2 – كلاس الخاص بالشخصيات

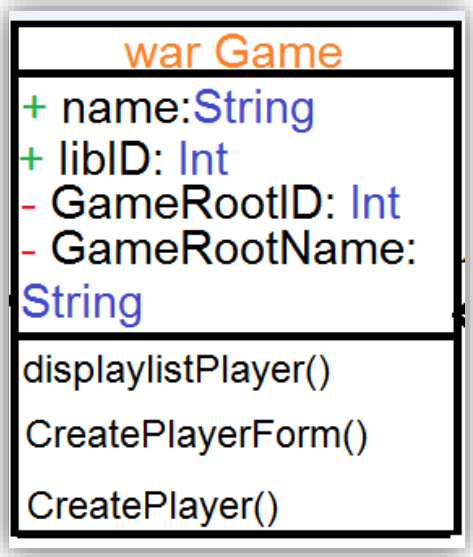
3 – كلاس خاص بالبيئة الخاصة باللعبة

هيا بنا نبدأ في التصميم :

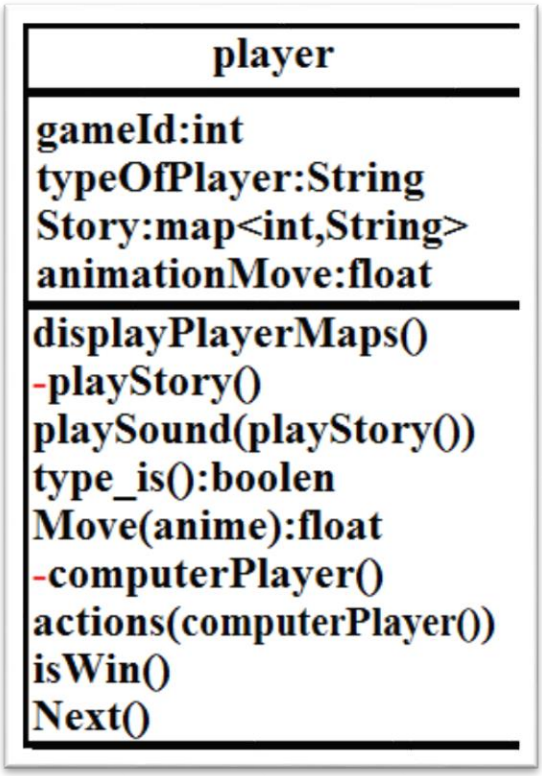
الخطوة الاولي : كلاس اللعبة

قمت بتصميم الكلاس بهذا الشكل الذي يفهمه الجميع وهو ما يسمي :

Class diagram

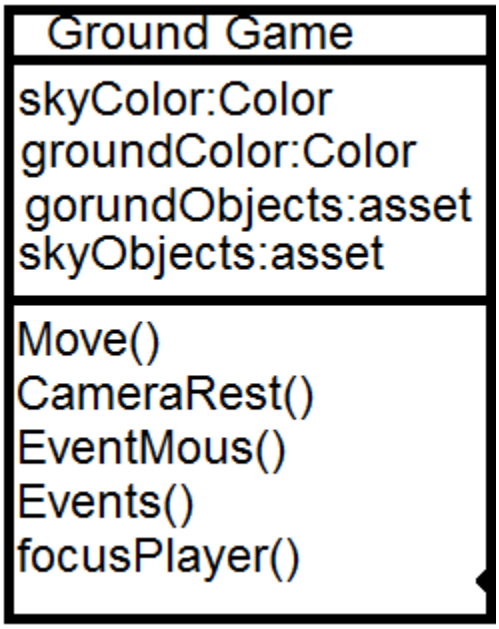


الخطوة الثانية : كلاس الخاص بالشخصيات



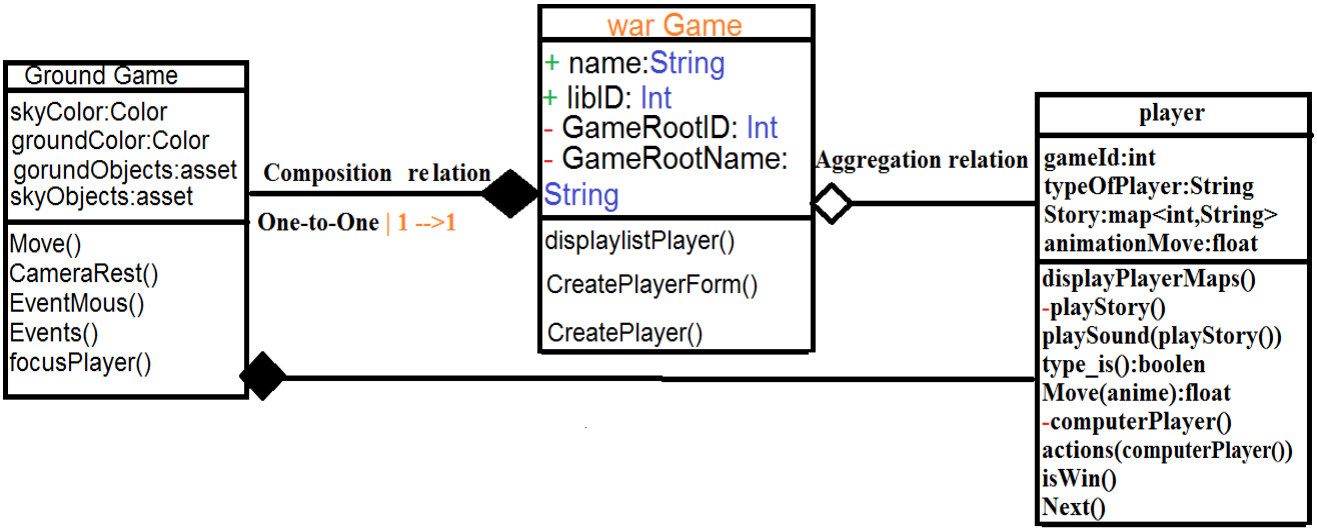
هو يمثل اللاعب والشخصية الخاصة به وميكانيكية تحريك الشخصية و القصة.

الخطوة الثالثة : كلاس خاص بالبيئة الخاصة باللعبة



وهذا التصميم يمثل البيئة التي تعيش بداخلها الشخصيات والكثير من الكائنات المتحركة والساكنة وبداخله أيضا دوال تتعلق بالتحكم في زاوية الرؤية و الاحداث التي تتم عند الضغط علي mouse or keyboard

في هذا الوقت يأتي دور صنع العلاقات وتكوينها بين الكلاس والأخر وهو الموضح في هذه الصورة .



CHAPTER 5

scope of class & Access modifier

1 – Scope ما هو

2 – Access modifier ما هو

ما هو Scope

هو نطاق عمل المكونات الداخلية للكلاس بحيث يمكن استخدامه لإثبات المكونات انها تابعة لهذا الكلاس .

امثلة علي كيفية طريقة الكتابة

(:: , . , ->)

مثال 1 :-

```
class test{
      int walk;
      string name;

      public:
            void set_walk(int);
            void set_name(string);
            void set_element_name(void);
            int get_walk(void);
            string get_walk(void);
            string get_element_name();
};
// class scope
// :: is pointer for scope
void test::set_walk(int w)
{
      this.walk = w;
}

....
```

مثال 2 :-

```
int main(){
      test object;
      // this shape is a scope (   .function_name() )
      object.set_walk(24);
      object.set_name("bosi");
      object.set_element_name("cat");


return 0;
}
```

مثال 3 :-

```
int main(){

      // scope of pointers to call members of class ( -> )
      test *objR;
      objR = object;
      cout << "the fast of walk is: " << objR->get_walk() << endl
      string name = objR->get_name();
      string nameOfElement = objR->get_element_name();
....

return 0;
}
```

ما هو Access modifier

هي مجموعة من كلمات رئيسية داخل لغة البرمجة تعبر عن تحديد إمكانية الوصول الي مكونات الكلاس فمنها :

- ❖ Public : هي كلمة رئيسية في اللغة تقوم بإظهار المكونات الرئيسية للكلاس لجميع النطاقات البرمجية الأخرى
- ❖ Private : هي كلمة رئيسية في اللغة تقوم بإخفاء جميع المكونات داخل الكلاس حتي لا يستطيع احد تغيير القيم الداخلية او التلاعب بها
- ❖ Protected : هي كلمة رئيسية في اللغة تقوم بإخفاء المكونات لدي الكلاس الا من يقوم بالوراثة من هذا الكلاس

CHAPTER 6

Inheritance

الوراثة : Single inheritance

الوراثة المتعددة : Multi-Inheritance

الوراثة المنفردة : Single inheritance

هو ان يكون لدي الكلاس الابن , كلاس والد واحد فقط يرث منه .

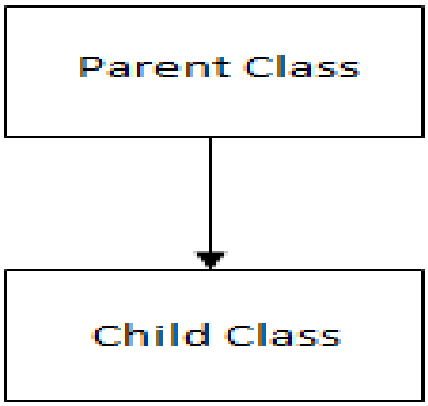


Fig: Single inheritance

كيفية الوراثة :-

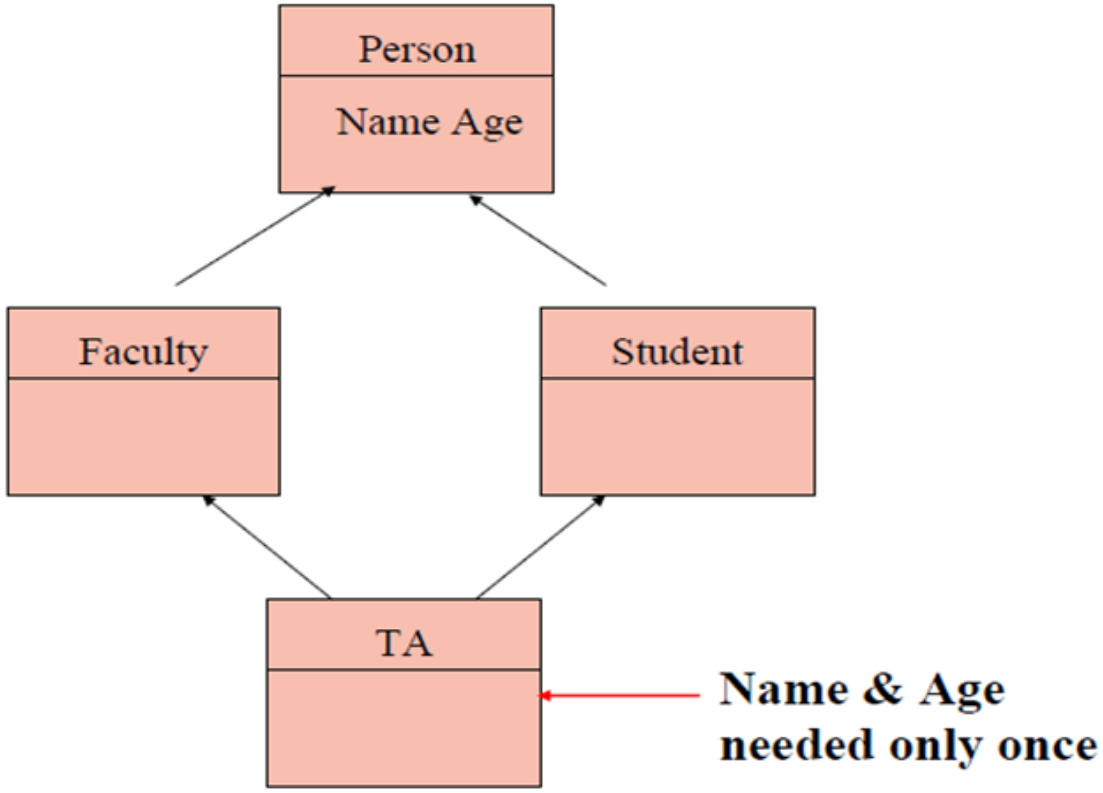
```
class staff
{
    private:
        char name[50];
        int code;
    public:
        void getdata();
        void display();
};

class typist: public staff
{
    private:
        int speed;
    public:
        void getdata();
        void display();
};
```

الوراثة المتعددة : Multi-inheritance

هو وراثة كلاس واحد من اكثر من كلاس .

وهي ميزة خاصة بلغة السي ++ وقد انتهت في بعض اللغات المنشأة علي لغة السي.



كيفية الاستخدام :-

```
class Person {
   // Data members of person
public:
    Person(int x)  { cout << "Person::Person(int ) called" << endl;   }
```

```
};

class Faculty : public Person {
   // data members of Faculty
public:
    Faculty(int x):Person(x)   {
       cout<<"Faculty::Faculty(int ) called"<< endl;
    }
};

class Student : public Person {
   // data members of Student
public:
    Student(int x):Person(x) {
        cout<<"Student::Student(int ) called"<< endl;
    }
};
 // multi-inherited
class TA : public Faculty, public Student  {
public:
    TA(int x):Student(x), Faculty(x)   {
        cout<<"TA::TA(int ) called"<< endl;
    }
};

int main()  {
    TA ta1(40);
return 0;
}
```

CHAPTER 7

Polymorphism

1 – تعدد اشكال الدوال (overloading methods)

2 – ما هي الدوال الوهمية (virtual method)

3 – كيفية تعدد اشكال الدوال الوهمية (override)

تعدد اشكال الدوال (overloading methods)

هو ان تقوم بتكرار دالة واحدة بنفس الاسم داخل كلاس واحد .

اذا سوف يأتي احد الان ويقول كيف اكرر الدالة اكثر من مرة بنفس الاسم سوف يقوم المترجم بإخراج أخطاء لان الاسم قد تكرر وبماذا استفيد من هذا التكرار .

أولا : يوجد قواعد محددة لتكرار الدالة بنفس الاسم وهذا ما يجعل إمكانية التكرار مسموحة بالنسبة للمترجم .

ثانيا : انت لا تقوم بتكرار الاسم فقط ولكن تريد ان تقوم بتكرار الاسم لعمل كذا حدث ولكن بنفس الدالة وهذا ما سوف نقوم بتوضيحه الان .

مثال : اذا كان لديك دالة تجمع رقمين واسمها Add

قد تريد جمع ثلاث ارقام في المرة الأخرى فيجب عليك عمل دالة اخري وتوجد لها اسم اخر وقد تريد جمع رقمين بصيغة عشرية فيجب ان تقوم بعمل دالة اخري وباسم اخر ولكن بهذا الأسلوب سوف يكون لديك الكثير من الدوال تقوم بنفس العمل تقريبا ولكن بأسماء مختلفة وهكذا يصعب عليك حفظ كل هذه الدوال التي صنعتها فما بالك بالمكاتب التي لم تصنعها ولهذا قد وفرت لنا لغات البرمجة هذه الميزة حتي لا نقع في الكثير من المشاكل .

قواعد الكتابة :-

1 – يجب ان تكون المعاملات Parameter متغيرة في العدد او في نوع البيانات

2 – تغيير نوع رجوع البيانات .

كيفية الكتابة :-

```
class printData {
   public:
      void print(int i) {
         cout << "Printing int: " << i << endl;
      }

      void print(double  f) {
         cout << "Printing float: " << f << endl;
      }

      void print(char* c) {
         cout << "Printing character: " << c << endl;
      }
};
```

ما هي الدوال الوهمية (virtual methods)

هي دوال يمكن إعادة تعريفها داخل الكلاس المشتقة من الكلاس المتواجد به الدالة الوهمية .

عن طريق وراثة من هذا الكلاس او الإشارة اليه بـ Reference

كيفية تعدد اشكال الدوال الوهمية:-

```
class Weapon
{
    public:
       virtual void loadFeatures()
         { cout << "Loading weapon features.\n"; }
};

class Bomb : public Weapon
{
    public:
```

```
       void loadFeatures()
         { cout << "Loading bomb features.\n"; }
};

class Gun : public Weapon
{
    public:
       void loadFeatures()
         { cout << "Loading gun features.\n"; }
};

int main()
{
    Weapon *w = new Weapon;
    Bomb *b = new Bomb;
    Gun *g = new Gun;

    w->loadfeatures();
    b->loadFeatures();
    g->loadFeatures();

    return 0;
}
```

CHAPTER 8

Advanced classes

الكلاس المتداخل (inner-classes)

Interface class

الكلاس المتداخل (inner-classes)

هو كلاس يعرف داخل كلاس اخر ويسمي : Nested class

يعرف علي اغلب الأوقات انه :-

Public

مثال :

```
class Host
{
public:
  class Nested
  {
  public:
    void PrintMe()
    {
      cout << "Printed!\n";
    }
  };
};

int main()
{
  Host::Nested n1;
  n1.PrintMe();

return 0;
}
```

ما هو Interface class

واجهة تصف سلوك أو قدرات الكلاس دون الالتزام بتنفيذ معين من هذا الصنف .

هو تعتمد علي بعض أنواع الدوال وهي تسمي : Pure function

مثال :-

```
// Base class
class Shape  {

public:
   // pure virtual function providing interface framework.
   virtual int getArea() = 0;

   void setWidth(int w) {
      width = w;
   }

   void setHeight(int h) {
      height = h;
   }

protected:
   int width;
   int height;
};

// Derived classes
class Rectangle: public Shape {

public:
   int getArea() {
      return (width * height);
   }
};

class Triangle: public Shape {

public:
   int getArea() {
      return (width * height)/2;
   }
};
```

CHAPTER 9

examples and Exercises

مثال علي (encapsulation)

مثال علي (operator overloading)

مثال علي (polymorphism)

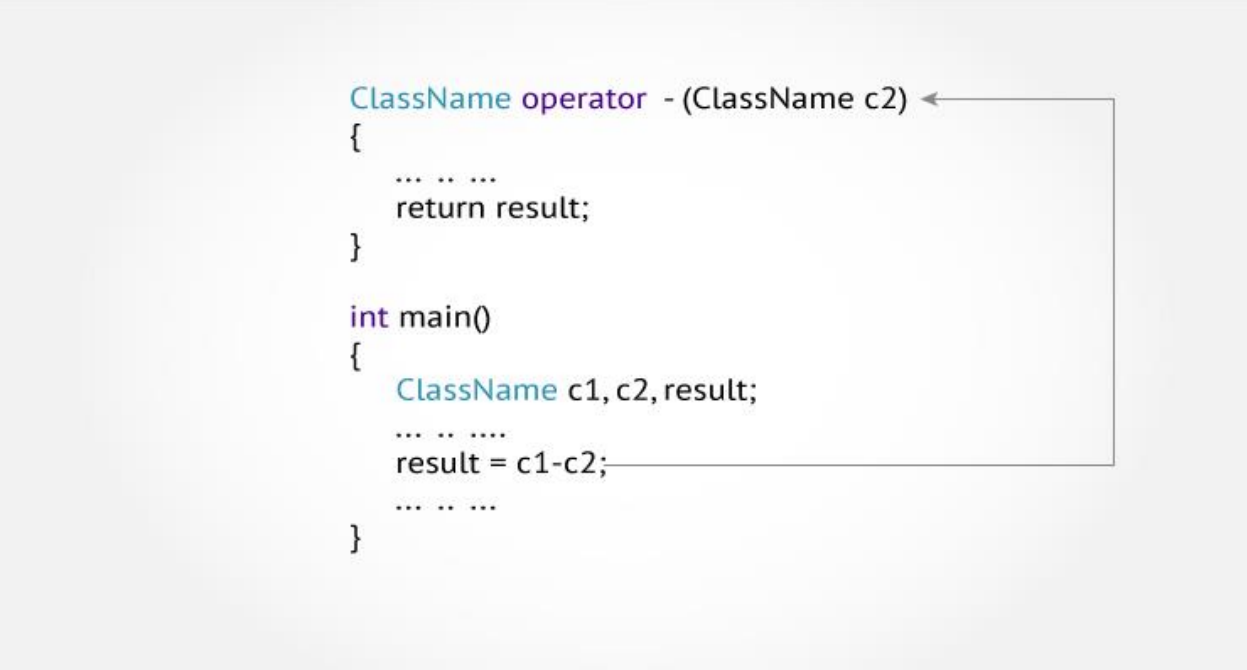
أسئلة علي البرمجة الكائنية OOP

مثال علي(encapsulation)

```
class Box {
   public:
      double getVolume(void) {
         return length * breadth * height;
      }

   private:
      double length;      // Length of a box
      double breadth;     // Breadth of a box
      double height;      // Height of a box
};
```

مثال علي (operator overloading)



مثال :-

```
class Test
{
   private:
      int count;

   public:
```

```
        Test(): count(5){}

        void operator ++()
        {
           count = count+1;
        }
        void Display() { cout<<"Count: "<<count; }
};

int main()
{
    Test t;
    // this calls "function void operator ++()" function
    ++t;
    t.Display();
    return 0;
}
```

مثال علي (polymorphism)

```
class Addition
{
public:
void sum(int a, int b)
{
cout<<a+b;
}
void sum(int a, int b, int c)
{
cout<<a+b+c;
}
};
void main()
{
clrscr();
Addition o;
o.sum(4, 2);
cout<<endl;
o.sum(14, 20, 50);
return 0;
}
```

أسئلة علي البرمجة الكائنية OOP

هنا في هذه النقطة سوف نتحدث عن اهم الأسئلة واكثرهم شيوعا في جميع لغات البرمجة فقط اذا اجبت علي هذه الأسئلة فانت قد اتممت جزء كبير واساسي في أي لغة برمجة لان الأسلوب واحد ولكن الصيغة قد تتغير .

ناسف اننا لم نستطع كتابة هذه الأسئلة مترجمة لأنها سوف تفقد المعاني الاصلية لها ولذا قد وفرناها لك باللغة الإنجليزية البسيطة التي يسهل فهمها وهناك فائدة اخري حينما تريد التقدم الي وظيفة وتم تحديد لك مقابلة كمهندس برمجيات لن يكن مرغوبا بك اذا كنت تريد ان تترجم كل هذه المصطلحات فالبعض ليس له ترجمة بنفس المعني .

1) What is a class ?
2) What is objects?
3) What is a pure virtual function?
4) What is a inheritance
5) How many type of inheritance?
6) What is a polymorphism ?
7) Abstraction class VS Interface class ?
8) What is encapsulation ?
9) What is a deferent between overloading / overriding ?
10) Make program do some operations like (* , / , - , +)
11) What is a constructor ?

CHAPTER Tips

What the next?

يمكن ان يكون لك الف وظيفة بعد ان تنتهي من اساسيات علم البرمجة الكائنية مثال هذا العمل هو :-

1 – تصنيع برمجيات سي ++ بكل انوعها فانت قادر علي ان تفهم أي مكتبة برمجية او ان تقوم بعمل مكتبة خاصة بك

2 – التعمق داخل اللغة وتطوير المكاتب البرمجية .

3 – عمل برامج اندرويد او ويندوز وبيعها علي المتاجر الخاصة بهم

4 – العمل علي بيئة تطوير QT

5 – دراسة لغات برمجية اخري مثل :-

(python , java , swift , kotlin , go , c# , etc..)

6 – التقدم الي شركة تصميم البرمجيات اذا توافرت بك الشروط التي تريدها الشركة

7 – تصميم الخوارزميات والذكاء الاصطناعي

8 – برمجة تطبيقات مفتوحة المصدر

9 – تصنيع تطبيقات للشبكات والتحكم بها IOT

10 – تدريب الطلاب الاخرين ...