12-12-2017

نظري

◄ دكتور المادة: سمير جعفر

◄ المحاضرة العاشرة والحادية عشر عنوان المحاضرة : محددات الوصول والوراثة

المستوى العلمي : أهلاً بكم أصدقائي في المحاضرة العاشرة من مقررنا الخوارزميات والبرمجة 2 سنتابع في الصفوف لنتحدث عن محددات الوصول ومن ثم ننتقل لمفهوم جديد وهو الوراثة.

محددات الوصول:

هي عبارة عن كلمات محجوزة في اللغة تحدد إمكانية الوصول والتعامل مع المكونات المحددة بها (داخل الصف) وتقسم الى ثلاث أنواع:

Public: محدد وصول عام والأعضاء التابعة لهذا المحدد يمكن الوصول اليها من أي مكان في البرنامج

Private:محدد وصول خاص والأعضاء التابعة لهذا المحدد لايمكن الوصول اليها الا من داخل الصف نفسه

Protected:محدد وصول محمي والأعضاء التابعة لهذا المحدد خاصة بالوراثة لايمكن الوصول اليها الا عن طريق الصف نفسه او صفوف مشتقة منه .

ملاحظة:

يمكن وجود اكثر من محدد وصول بنفس الصف –مجال المحدد هو كل شيء يندرج تحته حتى يظهر محدد اخر .

مثال توضيحي:

```
class A{
public:
int x;
private:
int y;};
```

بعد التصريح عن الصف A يوجد محدد وصول عام ومنه فكل مابعده من أعضاء يتبع له حتى ظهور محدد وصول اخر و بالتالي x فقط هو العضو العام

ومن ثم صرحنا عن محدد وصول خاص فكل مابعده يتبع له ومنه y عضو خاص

```
int main(){
A a = new A();
a.x = 2;
a.y = 5;
};
```

- الإعلان عن غرض a وبنائه
- Xعضو عام يمكن الوصول اليه من خارج الصف ومنه هذه التعليمة صحيحه
- Y عضو خاص لايمكن الوصول اليه من خارج الصف فهذه التعليمة خاطئه وتعطي خطا في البرنامج

ملاحظة:مايوجد داخل private لايمكن استدعائه داخل main.

مثال:

```
class A{public:
int x;
int y;
A(){x = 0; y = 0;}
void print (){cout << x << y;}
private:
int z;};
int main(){
A a = new A();
a.print ();
a.x = 2;
a.z = 5;
```

كل مايندرج تحت محدد الوصول العام تابع له يمكن الوصول الى الأعضاء والدوال واجراء عمليات عليها

كل مايندرج تحت محدد الوصول الخاص تابع له ولا يمكن الوصول الى الأعضاء والدوال واجراء عمليات عليها

Z عضو خاص لايمكن الوصول اليه واعطاؤه قيمة بالتالي التعليمه

البرمجة غرضية التوجه تقوم على ثلاث مبادئ :

1- التجريد (الصفوف والأغراض) ، يعني ذلك أننا نقوم بتمثيل النوع الجديد بشكل مجرد و بعد ذلك نستخدم هذا النوع الجديد من خلال الأغراض(أي المتحولات من هذا الصف)
2- الوراثة (موضوع محاضرتنا هذه) -3-تعددية الأشكال .

المبدأ الوراثي : يحاكي نوعاً ما الواقع إذ سيكون لدينا صفاً يرث من صفٍ آخر فيكون الصف المورث بمثابة أب، و الصف الوارث بمثابة ابن.

Class A صف ما (أب – قاعدة –مورث)مكون من ثلاث اقسام رئيسية:

private	protected	public
معطيات أعضاء طرق	معطيات أعضاء طرق	معطيات أعضاء طرق

وراثة من نوع معين
(محدد الوصول) "محمي – عام – خاص"

Class B صف جديد (ابن-مشتق-وارث) :

(جزء جديد) الابن			الموروث من الاب
عام	محمي	خاص	الجزء المحمي من الأب الجزء العام من الأب

ليكن لدينا الصف Aوليكن هو الصف الاب يحتوي على بيانات ومعلومات موجودة ضمن محددات الوصول وليكن لدينا صف اخر B وليكن هو الصف الابن تربط بينهما علاقة وراثة (يجب تحديد نوعها)

نلاحظ ان الصف B يتكون من قسمين :القسم الأول ماورثه من الصف A والقسم الثاني أشياء وبيانات جديدة لم ترد في الصف Aوخاصة بالصف B ونلاحظ ان الصف B لن يرث من Aالا البيانات الموجودة في المحددين publicوprotected بينما محدد الوصول private يحوي على بيانات خاصة بالصف Aولايمكن لاحد الوصول اليه.

- ان لغة c++ تسمح بالوراثة المتعدده أي يمكن ان يكون هناك اكثر من اب
- الصف الابن اكبر من الصف الاب وذلك لان الصف الابن يحوي الصف الاب إضافة لمكونات جديدة .
- يمكن ان يكون هناك اكثر من صف يرث من نفس الصف الاب .
- الشيء الخاص بالاب يبقى خاص ولايم توريثه ف ان الصف الابن يرث فقط المكونات العامة والمحمية .
- يرمز للوراثة برمجيا بالنقطتين (:) الصف على يسار النقطتين يرث من الصف الذي على اليمين .

- الأعضاء التي يرثها الصف الابن من الاب نعتبرها كشيء واحد وجميعها تكون تحت محدد وصول واحد أي اما تكون عامه او محميه او خاصه وذلك حسب نوع الوراثه.
- الوراثة تمكننا من التعديل على مايقوم به الصف الاب أي توسيع لعمل الصف الاب فنحن بالوراثة نحافظ على مايقوم به الصف الاب ونضيف شيء جديد ليقوم به
- ملاحظة:يحق للصف الابن إعادة تعريف ماوجد في الصف الاب .
- أي صف ابن هو صف اب والعكس غير صحيح
- هل يمكن اعتبار ان B هو A او ان Aهو B:

لتوضيح هذه الفكرة سنلجأ الى مثال المضلعات :

أي لو قلنا اننا نريد مضلع ولم نحدد أي شيء عنه فلن نعلم ماهو المضلع المطلوب وبالتالي سيكون الجواب هو المضلع بشكل عام

عند تحديد عدد اضلاعه ولنفرض انها أربعة اضلاع :سيكون الجواب هو المربع

ولو حددنا الاضلاع الثلاثة فسيكون مثلث و

هل كل مثلث مضلع ؟؟كل ابن هو اب وليس العكس صحيح (هل كل مضلع مثلث ؟؟؟)

- ويمكن للصف ان يكون له عدة أبناء وجميع الأبناء سيكون لديهم مايوجد عند الاب يمكن وجود ابن لاكثر من صف .
- **تعرف الوراثة برمجيا:**

Class اسم الصف الابن : نوع علاقة الوراثة اسم الصف الاب { مكونات الصف الابن};

كيف نقوم بكتابة وراثة في الكود البرمجي

$$\underbrace{class\ A}_{\text{الصف الأب}}\{\dots\dots\dots\};$$

$$\underbrace{class\ B}_{\text{الصف الابن}}\ :\ \underbrace{public}_{\text{نوع الوراثة يرث}}\ \underbrace{A}_{\text{الصف الأب}}\ \{\dots\dots\dots\};$$

$$class\ C: public\ B\{\dots\dots\dots\};$$

❖ **محددات الوصول في الصف الاب** تبقا كما هي اما في الصف الابن تكون محدودة حسب علاقة الوراثة :

الوراثة عامة فان جميع المكونات التي تنتقل من الصف الاب الى الصف الابن تكون عامة

الوراثة خاصة فان جميع المكونات التي تنتقل الى الصف الابن تكون مكونات خاصة

الوراثة محمية فان جميع المكونات التي تنتقل الى الصف الابن مكونات محمية

❖ عندما نريد استخدام الصف الابن يجب ان ننشئ غرض من هذا الصف لذلك يجب ان يكون له باني وبما انه موروث فانه يجب علينا ان نضمن بناء هذا الغرض من الصف الاب لان قسم من الغرض من الصف الابن وقسم من الصف الاب وهنا يجب على باني الابن ان يستدعي باني الاب ولذلك علينا ان تكون اول تعليمة من باني الصف الابن هي استدعاء لباني الصف الاب واذا لم تكتب هذه التعليمة فان اللغة ستقوم باستدعاء الباني الخالي (اذا لم يوجد أي باني بالصف الاب)

```
class A{ public :
int x;
A(int x){this ->x=x};};
class B:public A{
double y;
B( ){y=0;}
```

في التمرين السابق ورد لدينا خطأ وهو عدم وجود الباني الخالي في الصف الاب ولحل هذه المشكلة يجب كتابته في الصف الاب لالنه اذا لم نضع باني خالي في الصف الاب وعند باني الابن لم نضع استدعاء لباني الاب فهذا يعطي خطا (ولأن لاوجود لباني خالي و اللغة لن تولد باني افتراضي لوجود باني مكتوب في الصف اب يحوي وسطاء وليس خالي في الصف الاب) . والحل الصحيح :

```
class A{ public :
int x;
A(int x){this ->x=x};};
class B:public A{
double y;
```

إن هذا السطر يعني أن الصف B سيرث كل ما هو عام و محمي في الصف A و قد حددنا أن المعلومات الموروثة ستكون في B عامة و ذلك عندما قلنا أن نوع الوراثة public

```
B():A(0){y=0;}
B(int x ):A(x){y=0;}
B(int x,double y):A(x){this ->y=y;}};
```

مثال اخر:

```
class A{public :
int x;
A(int x){this ->x=x;};
void print ( ){cout<<x;}};
class B:public A{public:
double y;
B( ):A(0){y=0;}
};
int main(){
B.b=new B( );
b.print();
return 0;}
```

استدعاء للباني

استدعاء دالة من الصف الاب

اذن سيطبع قيمة x وهي (0).

اما هنا في هذا المثال :

```
class A{public :
int x;
A(int x){this ->x=x;};
```

```
void print ( ){cout<<x;}};
class B:public A{public:
double y;
B( ):A(0){y=0;}
void print ( ){cout<<"coco";}
};
int main(){
B.b=new B( );
b.print();
return 0;}
```

دالة الصف الابن

في هذه الحالة سيقوم بطباعة coco أي ان الدالة التي استدعيناها في main باسم الصف الابن اذا كانت معرفة في الصف الابن فسيقوم بتنفيذ ماطلب منها في الصف الابن بينما لو لم تكن معرفة او موجودة في الصف الابن أصلا فاللغة ستفهم انه بعلاقة وراثة مع الصف الذي قبله لذلك ستنفذ ماطلب منها في الصف السابق (الصف الاب).

المكدس : (*stack*)

هو عبارة عن قائمة خطية تم تقييد إضافة عنصر و اخراج عنصر منها حيث نقوم بإضافة وحذف العناصر من رأس القائمة فقط . ويحوي المكدس دائما على مؤشر يشير الى اخر عنصر موجود في المكدس

ويرمز لها بـ : *last in first out (LIFO)*:

$$struct\ node\ \{$$

$$int\ val;$$

$$node * next;$$

$$\};$$

$node * q;$ قائمة خطية جديدة

$$node * x = new\ node;$$

$$x \to val = 5;$$

$$x \to next = NULL;$$

$$q = x;$$

$$x = new\ node;$$

$$x \to val = 10;$$

$$x \to next = q;$$

$$q = x;$$

و لحذف العنصر الأول :

$$q = q \to next;$$

الرتل : (*Queue*)

هو عبارة عن قائمة خطية تحقق ما يلي :

أول عنصر يدخل هو أول عنصر يخرج وبالتالي الاضافة تتم بنهاية الرتل (القائمة) والحذف يتم من البداية (رأس القائمة).

ويرمز لها :*First in first out (FIFo)* و بالاستفادة مما سبق يتم الإضافة على الذيل (*tail*) والحذف يتم من الراس (*head*) أي بداية القائمة.

إلى هنا أصدقائي يكون قد انتهى مقررنا .....

نتمنى أن نكون قد قدمنا مادة مفيدة نالت اعجابكم ...

مع تمنياتنا بالتوفيق الى جميع الزملاء في جميع موادهم ...

انتهى المقرر

إعداد: عبير خزنة كاتبي وسندس درويش ووفاء شيخ سالم