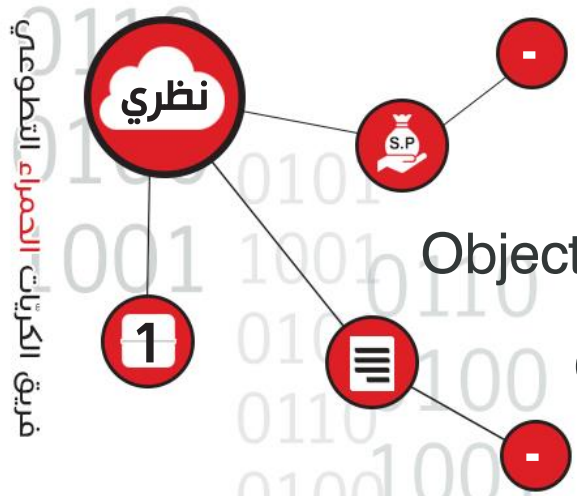


مفاهيم أساسية في الـ Object Oriented Programming

د. محي الدين مراد

RB Informatics; 2 / 10 / 2019
لغات البرمجة

تمهيد:

تحدثنا في السنوات السابقة عن Structure Language (لغات بنيوية أو اجرائية تستخدم لحل مشكلة لها طابع تسلسلي Sequential يتم حلها بمجموعة من العمليات المتتالية) و تطورت البرمجة لتواكب المشاكل الحياتية.

و إذا نظرنا بأكثر تجرداً لواقع الحياة سنجد أنها عبارة عن تفاعل وتواصل Objects مع بعضها البعض، تتواجد في الـ Runtime (وقت العمل) الذي يتم فيه فيه حل المشكلة (أو القيام بأي مهمة).

أما بالنسبة لبنية (structure) هذا الـ Object والتي يتم تعريفها على أنها الـ Class فتتواجد أثناء العمل على التصميم الذي سوف يحل هذه المشكلة (DesignTime).

سنتحدث بداية عن مجموعة من المفاهيم والتي سيتم عليها بناء المقرر:

- Inner Class: وهي Classes موجودة ضمن Scope لـ Class آخر.
- Composition (التركيب): والتي تندرج ضمن الـ class reuse أي القيام باستخدم objects من classes أخرى لتركيب الـ composed class وتكون هذه الـ objects من classes غير مرتبطة بالـ composed class لذا يجب التفرقة بين مفهومي الـ inner class والـ composed class.
- Class diagram (مخطط الصفوف): وهو الذي يحدد الصفوف التي يتم استعمالها لحل المشكلة وآلية ارتباطها مع بعضها و يحدد مهمة وسبب وجود كل صف ضمن البرنامج (يتم استخدام لغة UML: Unified Modeling Language سنتعرف عليه بشكل أكبر ضمن مادة هندسة البرمجيات).
- Exception (الاستثناء): وهو عبارة عن حدوث خطأ غير مسيطر عليه يسبب الخروج من البرنامج مما يؤدي إلى عدم اداء البرنامج لمهمته بشكل صحيح، اي خروج البرنامج إلى نقطة مخالفة للنقطة التي تم تحديدها للبرنامج ليتم الخروج وأداء المهمة بشكل صحيح.
- Multi-Threading: هو عبارة عن القدرة على استخدام موارد الآلة التي يتم تنفيذ البرنامج عليها لتنفيذ أكثر من عملية بنفس الوقت والتي تخدم نفس المهمة.

مثال: بفرض لدينا مصفوفة تحوي على مليون خانة ونريد تعبئة هذه الخانات عند التنفيذ بشكل أحداي (Single-Threaded performance) فإن الوقت الذي المُتطلب لتنفيذ هذه المهمة هي عبارة عن مليون وحدة زمنية، أما في حالة الـ multi-threaded performance وبفرض لدينا آلة لها القدرة على تنفيذ 4 threads فإن اول thread سوف يقوم بتعبئة الخانات من 0 لـ 250 ألف وبنفس الوقت سوف يقوم الـ thread الثاني بتعبئة الخانات من 251 ألف لـ 500 ألف وهكذا.... عندها سوف ينخفض وقت التنفيذ لربع ما كان عليه بسبب وجود 4 thread تتعاون للقيام بنفس المهمة.

ملاحظة:
يجب أن تكون الآلة متوافقة من ناحية الـ hardware من أجل القدرة على التنفيذ على التوازي.

- **Collection:** هي عبارة عن بنى معطيات يتم التخزين فيها مجموعة من ال objects مثل (list, stack, array, queue,....) و مثال على ذلك : بفرض لدينا array of persons يمكننا جمع كل الـ objects التي تندرج ضمن إطار الـ person (ترث من person) مثل الـ .....student, professor, fireman، و وضعها ضمن ال array . وسنتحدث لاحقا عن أنماط أخرى من الـ collection.

ملاحظة: إن كل من لغتي C/C++ هي عبارة عن لغة إجرائية (تسمح C++ بتطبيق مفهوم الـ OOP إلى جانب المفهوم الإجرائي و بذلك فهي تسمى بـ hybrid language)، ولكن لغة Java أو C# هي عبارة عن Pure OOP language أي انها لاتسمح بأي behavior أو أي Data أن تكون خارج إطار class اي أن تكون كل الـ functions والـ data members مغلفة (Encapsulated) ضمن classes.

بنية برنامج OOP languages:

```
Class Prog {
        public static void main() {
        }
}
```

وهو يمثل الـ **Start point** التي قمنا بتحديدها للبرنامج.

يتصف الـ **access specifier** الخاص به على أنه **public** وذلك لإنه اذا كان **protected** أو **private** لن يكون للـ **compiler** القدرة على رؤية هذا التابع والقيام بتنفيذه.

ويتصف ايضا أنه **static** وذلك لأنه عند تشغيل البرنامج لن يكون هناك **instance/object** من الـ **class** موجودة لتنفيذ هذا التابع لذا يجب أن يكون **static**.

ملحوظة ☺:

يمكن أن يكون هناك أكثر من تابع main ضمن نفس الـ class وهي حالة عمل overloading.

يمكن أن يكون اكثر من تابع main في اكثر من class ولكن عندها يجب تحديد تابع الـ main الذي سوف يبدأ به البرنامج

يمكن لتابع main أن يقوم باستدعاء تابع main اخر في class اخر.

فنلاحظ أن تابع الـ main هو تابع مثل اي تابع اخر ولكن مهمته هي إرشاد الـ compiler إلى نقطة بداية البرنامج.

ليكن لدينا الـ classes التالية:

```
Class Prog {
        Public static void main() {
                Std s1 = new Std();
        }
}
class Std {
        Public Int id;
        Public static String univ;
        public static func1() { }
        public func2() { }
}
```

وهنا يمثل التابع الباني لـ object من نوع Std. ☺

Data Members

Methods

يجب أن نلاحظ نوعين من الـ Data members حسب المثال السابق:

Object data members: وهي عبارة عن data member خاص بالـ object يتم حجزه لكل object بشكل خاص عند إنشاء instance من هذا الـ class مثال على ذلك id من الـ class Std.

Class data members: وهي عبارة عن data member خاص بالـ class يتم حجزه و مشاركته مع كل الـ objects من نفس النوع، اي عندما يقوم احد الـ objects بالتعديل على قيمة هذا الـ data member سوف يتم تعديله بشكل عام عند كل الـ objects (أي أن كل الـ objects سوف ترى هذا التعديل على القيمة) مثال على ذلك String static univ من class Std.

ملاحظة: إن الـ object data member يتم تهيئته ضمن الـ constructor الخاص بالـ class وكذلك الـ class data member ولكن عندها لن يتم تهيئة هذا الـ data member إلا عند عمل instance من هذا الـ class، وقد تكون المسألة لا تتطلب إلا التعامل مع الـ class data member لذا ليس من الصحيح دوما الاعتماد على الـ constructor لتهيئة الـ class data members.

ونلاحظ أيضا نوعين من الـ methods:

class methods: وهي عبارة عن توابع لا تحتاج إلى عمل instance من الـ class لاستدعائها، ولكن هذه التوابع لا يمكن لها أن تتعامل إلا مع class data members حصرا وذلك لأنه لا يمكن أن يتعامل مع data لم يتم حجزها بعد وذلك بسبب عدم ضمان إنشاء instance من هذا الـ class تحوي على الـ object data members التي تم تعريفها ضمنه، ولكن ذلك لا يمنع إنشاء objects بداخله فهو بالنهاية تابع مثل اي تابع اخر.

object methods: وهي عبارة عن توابع تحتاج إلى عمل instance من الـ class لاستدعائها، ويمكنها أن تتعامل مع كلا object and class data members.

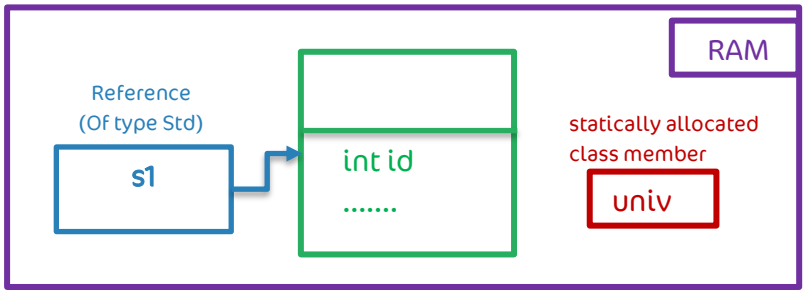
إنشاء الـ Objects:

إن اللغات غرضية التوجه مثل Java لا تعتمد على المؤشرات في آلية الحجز بل تعتمد بشك مباشر على مواقع بالذاكرة reference كل منها يكون مُهيأ بحسب البرنامج ليقوم بالإشارة أو أن يحتوي على نوع معين من الـ data، ليكن لدينا المثال التالي الذي يحوي على الـ classes التالية:

```
Class Std  {
    Int id;
    Public String static univ;
    public static func1() { }
    public func2() { }
}
```

```
Class Prog {
    Public static void main() {
        Std s1;
        s1 = new Std();
    }
}
```

عند بداية البرنامج السابق نلاحظ أنه سوف يتم مباشرة حجز موقع بالذاكرة للمتحول univ وذلك لأنه class data member (static) ، وعند تنفيذ أول تعليمة ضمن التابع main سوف يتم تهيئة موقع في الذاكرة ليتم حجز object فيه من نوع Std، ثم يتم حجز هذا الـ object عند تنفيذ التعليمة التالية.



مثال على مسألة لها طابع OOP:

ليكن لدينا class طالب و class سيارة، وبفرض أن كل طالب لديه سيارة ويقوم بتشغيل هذه السيارة بشكل دوري أي انه يوجد behavior هو عبارة عن start engine، فيكون هذا الـ method/behavior موجود ضمن class السيارة ويقوم الطالب باستدعاء هذا الـ behavior بعد إنشاء instance من سيارته وذلك بغرض تشغيل السيارة، ولكن يجب أن يكون تابع تشغيل السيارة هو object method وذلك تجنباً لتشغيل كل السيارات (class method يقوم بالتعديل حصرا على class data members وبالتالي سوف يتم التعديل على كل الـ objects).

- Class reuse:

وتتمثل بالـ composition والوراثة (inheritance) ويتم تطبيق مبدأ الوراثة عند ايجياد مجموعة من الـ data members أو الـ behaviors المشتركة بين الـ objects فيتم إنشاء class أب يحوي على هذه القواسم المشتركة.

في أغلب الأحيان يكون الـ class الأب له طابع التجريد أي أن يكون abstract class لا يمكن إنشاء objects منه ويحوي على constructor (تذكر لا يتم استدعاء constructor الأبن بدون استدعاء constructor الأب) بالأضافة إلى الـ data members and functions.

تعددية الأشكال polymorphism:

تعددية الأشكال، وهي خاصيّة أن يأخذ الغرض أكثر من شكل. أبرز مثال على ذلك: في الوراثة، عندما يشير reference من class الأب إلى object من class الابن.

مثال: ليكن لدينا الـ classes التالية:

```
class Pers {
        int id;
        String name;
        public void k();
}
```

```
Class Std extends Pers {
        int a;
        string univ;
}
```

```
class Main {
        public static void main() {
                Pers  s1 = new Std();
                s1.k();
        }
}
```

في البداية سوف يتم إنشاء reference من نوع Pers ومن ثم إنشاء Object من نوع Std وربطه معه، يتم بعدها استدعاء التابع k() فنلاحظ أن التابع k() غير موجود ضمن Std ولكنه موجود ضمن class اب وهو Pers عندها سوف يحدث ما يسمى بالـ later binding والذي يحدث خلال الـ runtime ليصبح الـ s1 مربوط مع object من نوع Pers ليتم عندها تنفيذ التابع k().

ملاحظة:

في المثال السابق أثناء الـ compile time يكون التابع k() مربوط بـ Pers وذلك لأنه في الـ compile time يتم الاعتماد على نوع الـ reference بالربط (نوع الـ reference هو Pers)، أما خلال الـ runtime يحدث ما يدعى بالـ later Binding ليصبح reference s1 يشير إلى object من نوع Std عندها يتم ربط التابع k() بـ Std وذلك لأنه يتم الاعتماد على نوع الـ object بالربط خلال الـ runtime.

يمكن تعريف الـ later binding كالتالي: وهي تقنية يتم من خلالها تحديد الـ method التي يتم استدعاءها من أجل object محدد خلال الـ runtime.

مثال: ليكن لدينا الصفوف التالية:

```
abstract class Shape{
abstract void draw() ;
abstract void move() ;
abstract void clear() ;
}
```

Abstract Class

```
class circle extends Shape{
        override void draw() { }
        override void move() { }
        override void clear() { }
}
```

```
class Do{
        public static void MoveShape(Shape s){
                s.draw()
                s.move()
                s.clear()
        }
        public static void Main(){
                Shape c1 = new circle();
                MoveShape(c1);
        }
}
```

```
class Triangle extends Shape {
        override void draw() { }
        override void move() { }
        override void clear() { }
}
```

بداية قمنا بتعريف class Shape وهو عبارة عن abstract class لا يمكن إنشاء objects منه، ثم ثمنا بتوريث هذا الـ class لكل من Circle و Triangle وقمنا بعمل override لكل التوابع الخاصة بـ Shape لان التوابع هي من نوع abstract أي من الضروري عمل override لها

بعده قمنا بإنشاء class Do يحوي على تابع static MoveShape له وسيط هو عبارة عن reference لـ Shape، ثم ضمن تابع الـ Main قمنا بخلق object من نوع Circle واستدعينا التابع MoveShape على هذا الـ object عندها سوف يتحقق مفهوم تعددية الأشكال مما يؤدي إلى استبدال الـ object s بالـ object c1 (يبقى الـ reference Shape) ويتم استدعاء التوابع draw, move, clear الخاصة بـ c1 لأن الربط ضمن الـ runtime يعتمد على نوع الـobject وليس الـ reference.

ملاحظة :

abstract function في لغة جافا هو نفسه pure function في لغة c++.

أي يجب عمل override له ضمن class الابن.

انتهت المحاضرة .....