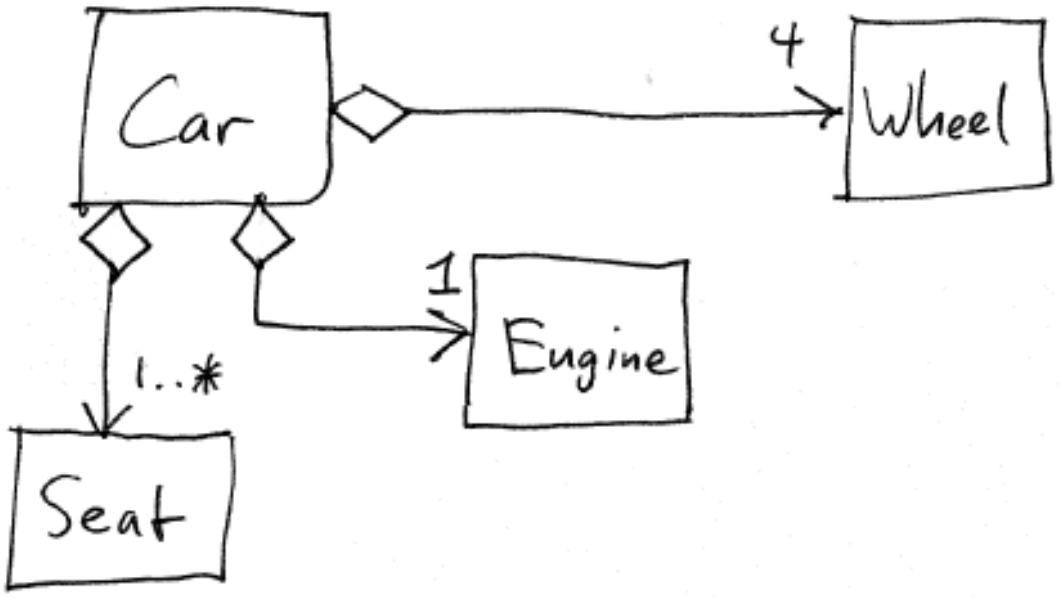
بواسطة وجدي عصام | 27/09/2013 | هندسة برمجيات | 5 تعليقات | 1،375 views

ما الفرق بين ال Composition و ال Aggregation



عالم البرمجه له كثير من الأوجه أو الأقسام programming paradigm منها البرمجه الإجرائيه procedural programming مثل لغه السي والباسكال (البعض يطلق عليهم imperative language) ، نوع أخر وهي الFunctional programming وهي تختلف عن الإجرائيه مثل لغه ليسب فالبرمجه هنا تستخدم مفهوم الlist بشكل كبير وتختلف طريقه كتابه البرامج عن الطريقه المعروفه بالاضافه الى وجود بعض نقاط تشابه بين هذا النوع والimperative ،، وجه أخر من أنواع البرمجه وهي البرمجه الكائنيه object oriented programming وهو يعتبر من أفضل الparadigm وخاصه في البرمجيات التجاريه والسبب أن استخدام هذا الأسلوب يسهل في حل المشكله بنسبه كبيره لأننا سنحاول تقليد العالم الحقيقي من خلال كائنات تشابه تلك الكائنات التي نحاول وضع حل لمشكله لها ..

بشكل عام كيفيه استخدام البرمجه OO واستخدام المفاهيم الرئيسيه مثل الوراثه inheritance وتعدد الأوجه polymorphism والعموميه generic والكبسله وغيرها أمر ليس بتلك الصعوبه ويستطيع المبتدئ تعلمها في غضون أيام ،وتبقى المشكله في كيفيه معرفه متى يمكن أن تستخدم الشيء القلاني هو الموضوع الصعب ، مثلا كيف تقوم بعمل كلاس عمليه بسيطه في أي لغه ، ولكن مثلا كم كلاس ستقوم لبرمجه نظام Bank Account ، وما هي نوعيه العلاقات بين هذه الكلاسات ، ومن هو الكلاس الرئيسي الذي ينشئ البقيه ، ولماذا الكلاس الأخر يحتوي على متغير من هذا النوع ، هل يمكن جعل هذا المتغير كلاس قائم بذاته ووو الكثير من الأمور وهي مختصه بموضوع التحليل والتصميم الكائني Object Oriented Analysis & design.

بعد تمكنك من فهم أساسيات موضوع OOAD فتكون أنجزت نصف الطريق ،، فما زلت عند طلب برنامج متكامل ستواجهه صعوبه في اتباع طريقه منطقيه لكل تبدأ من البدايه في كتابه البرنامج الى النهايه وبشكل ملائم لمتطلبات العميل وفي الوقت المحدد in deadline والcost المناسب .. وهنا يكمن أهميه الProcess أو الMethodology المستخدمه لتطوير البرنامج وهناك الكثير من المنهجيات ودائما في عالم الOO نفضل الincrement-iterative لأنها تسمح للعميل بأن يغير متطلباته بلا مشاكل ، وفي كل دوره بسيطه في المشروع -قد تكون من اسبوعين الى شهرين- نقوم بعرض اصدار جديد من المشروع يحتوي على ميزه جديده ، وهكذا في كل دوره يكبر المشروع الى وقت التسليم ،، وكل هذا بمشاهده العميل ومرافقته لكل الأحداث وبالتالي نضمن عدم رفض العميل للمشروع أو احداث تغيير كبير جدا لأنه كان مرافق لعمليه التطوير ،،

هناك الكثير من الprocess المستخدمه مثل RUP و eXtreme Programming وهي أحد أبناء الAgile وهناك الripple وغيرها من الطرق والمنهجيات والتي يفضل أن تستخدم كل واحده في حاله من الحالات تكون هي الأفضل عن غيرها ،، سوف نتحدث قريبا عن هذا الموضوع بشكل موسع للغايه ان شاء الله بموضوع Object Oriented Software development ، وحاليا سنركز على جزئيه الكائنات ونصف أحد أهم العلاقات بين الكائنات وهي العلاقه Composition و العلاقه aggregation ونعرف الفرق بينهم ،، ونترك بقيه العلاقات (الوراثه inheritance و Association) والتفاعل بين الكائنات للموضوع الذي سنطرح فيما بعد بمشيئه الله ..

في البدايه وبشكل عام ، لا يوجد فرق كبير من الComposition والAggregation حيث يعني هذا النوع من العلاقات بعلاقه أحتواء أو علاقه Has-A بمعنى كائن يحتوي على كائن ،، لنأخذ مثال السياره تحتوي على عده كائنات مثلا محرك السياره والماكينه ودواسه الوقود ووو كل هذه المكونات مع بعض تكون ما يسمى بالسياره .. أي أن السياره "تحتوي" على جميع هذه المكونات ..

مثال أخر جهاز الحاسب Computer يحتوي على عده مكونات وهي الذاكره RAM والمعالج CPU ومزود الطاقه Power Supply واللوحه الأم motherboard ووو غيرها من المكونات ،،

وبالطبع بما أن لكل مكون من هذه المكونات خصائص وسمات خاصه به فأننا لن نستطيع استخدام متغيرات عاديه primitive data type لوصف هذه المكونات ، أي أننا لا تستطيع أن نقوم بعمل متغير int لكي يمثل الذاكره أو المعالج لأن لكل من هذه المكونات وظيفه خاصه فيه ومتغيرات Internal بها خصائص هذا المكون ،، وبالتالى سيكون أي مكون من هذه المكونات عباره عن كائن من كلاس يمثل Model هذا المفهوم .. (ملاحظه : الكلاس هو Model فقط أو عباره عن قالب blue-print ومن خلاله نقوم بعمل كائنات Objects من هذا الكلاس .)

ملاحظه مهمه /
ابتعد تماما عن ترجمه أي مصطلح مهما كان حتى لا تدخل في مشاكل في الفهم أنت في غني عنها ،، فمثلا سوف تجد في الكتب من يقول أن علاقه compoisition هي has-a وستجد من يقول هي use-a وستجد وستجد .. لذلك دع جميع هذه المصطلحات وقم بفهم العلاقه فقط وقم بصياغتها بطريقتك الخاصه ..

بنفس الأمر هناك من يفرق بين الobject والobject instance !! مع أن العبارتين تعني عمل كائن من الكلاس ولا يوجد أي فرق بينهم .. ولكن محاوله الترجمه الحرفيه كما ذكرت سيدخلك في مشاكل أنت في غني عنها ،،

نعود للComposition والAggregation ،، كما ذكرنا أنهم علاقه أحتواء (كائن يحتوي على كائن) فقط .. لكن هناك فرق بسيط بينهم (لدرجه أن هناك كتب ومقالات تتجاهل هذا الفرق عندما يتحدث عن التصميم بشكل High level ، لأنه فرق بسيط جدا)..

الفرق هو أن الكلاس الذي يحتوي على بقيه الكائناتComposition هو المسؤول عن عمليه انشاء هذه الكائنات وعمليه انهائها ، ونقصد هنا هو المسؤول عن حجز مواقع لهذه الكائنات في الذاكره ، وهو المسؤول عن تحرريها ، بمعني أوضح أن الكلاس الكبير بمجرد انشائه سيتم انشاء جميع المكونات (الكائنات) بداخله ،، وبمجرد أنتهاء الكلاس الكبير سيتم فورا انهاء جميع الكائنات بداخله ،، لنأخذ مثال السياره لكي نوضح القصد ، هل يمكن أن تصبح السياره سياره بعد أخذ المحرك منها أو الماكينه ؟ بالطبع لأ .. اذا الكلاس الكبير لن يتم انشائه الا اذا تمت انشاء جميع ما يحتويه من كائنات .. لذلك يطلق على علاقه Composition بعلاقه قويه Strong Relationship .

النوع الأخر وهو الAggregation وهو أيضا علاقه احتواء ولكن في هذه الحاله الكلاس الكبير هو غير مسؤول عن انشاء الكائنات (أي حجز موقع في الذاكره لها) وغير مسؤول عن عن تحرريها ،، اضافه الى أنه تم انهاء الكائن الكبير فهذا قد لا يؤثر على المكونات بداخله وقت تعمل بعد ذلك ،، مثلا قسم في جامعه يحتوي على عدد من المدرسين ، في حال تم نقل القسم أو الغائه فلن يتم الغاء المدرسين الذين عملوا بالقسم بل سيكونوا موجودين وربما ينتقلوا لقسم أخر أو جامعه أخرى ..

الى هنا وقد يكون الفرق غير واضح بينهم ، لكن بقليل من الكود سيتضح المقصود بذلك ، وسوف تتفاجئ بأنك كنت تستخدم أحد النوعين بكثره في برامجك ، وربما كنت تستخدم طريقه hybrid أي تستخدمهم مع بعض في نفس الوقت !
نبدأ بقليل من الكود لكي نوضح الفكره ، بالاضافه لتوضيح كيف يتم رسم هذه العلاقات باستخدام UML .

لنبدا الcomposition ، كما ذكرنا أن الكائن هو المسؤول عن جميع الكائنات بداخله وهو يتولى عمليه انشائها وانهائها ، بالتالي تكون هذه الكائنات موجوده لديه ، مثلا لنعد لعلاقه المحرك والسياره وننظر لمقطع الكود التالي (قد نستخدم في بقيه الأمثله نموذج خليط بين كود سي++ وجافا ، فالغرض من الموضوع توضيح النظريه والفكره وليس التطبيق الفعلي) :

Java

```
1
2  public class Engine
3  {
4  . . .
5  }
6
7  public class Car
8  {
9      Engine e = new Engine();
10     .......
11 }
12
```

Figure 1 - Composition

بالنسبه للaggregation ، نأخذ مثال شخص person مع عنوانه ،، وسنجد أنه بانتهاء هذا الشخص فأن العنوان موجود وربما نقوم باعطائه لشخص أخر فيما بعد ،، ننظر للمقطع التالي :

Java

```
1
2  public class Address
3  {
4  . . .
5  }
6
7  public class Person
8  {
9       private Address address;
10
11      public Person(Address address)
12      {
13          this.address = address;
14      }
15      . . .
16 }
17
18 public static void main (string args[] ) {
19 Address address = new Address();
20 {
21 Person person = new Person(address);
22 }  // say here person delete or end of its scope
23
24 // address still can be used here
25
26 }
27
```

بالنظر للمثال أعلاه سنجد أن بعد عمل person ، وانتهائه (لأنه خرج من الscope الذي عرف فيه) أن العنوان ما زال موجودا ويمكن أستخدامه مره ،، وهو ما يعرف بaggregation . وفي UML يكون بمعين لكن بدون لون ويدل على خفه العلاقه weak relationship ...

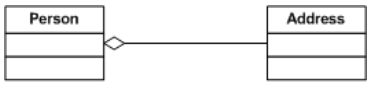
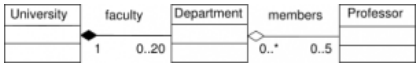


Figure 2 - Aggregation

مثال أخر لكي تتضح الفكره :

Java

```
1
2  class Professor;
3
4  class Department
5  {
6    ...
7    private:
8      // Aggregation
9      Professor* members[5];
10   ...
11 };
12
13 class University
14 {
15   ...
16   private:
17     // Composition
18     Department faculty[20];
19   ...
20 };
21
```



أخيرا ستجد كم من المصطلحات لوصف هذه العلاقات ولكل منها معني قد يشوشك في حاله الترجمه ، فمن يقول بأن الcomposition يسمى Composition / Composition Aggregation ، و Aggregation يسمى Shareable Aggregation . وهناك مسمى بأن الcomposition هو Aggregation by value (والسبب أن الكلاس الكبير قد يأخذ قيمه value ويضعها في المكونات ولكنه هو المسؤول عن انشائهم وكانها شبيه بالتمرير بالقيمه) ، والAggregation هو Aggregation by reference (بمعني أن القيمه المرره للكلاس الكبير للمكونات هي موقع في الذاكره reference وأنه فقط سجعل تلك المكونات تؤشر لهذه القيم بالذاكره) .، ولكنها في النهايه هي مصطلحات من منظور شخص معين ، يمكنك بعد قرائتك للدرس وفهمك أن تخترع مصطلحاتك الخاصه اذا أردت .. لذلك تعامل مع المفهوم والعلاقه بشكل مبسط واترك ترجمه المصطلحات فقط بكل بساطه :).

للمزيد :

Object composition

Aggregation

Composition

Aggregation versusComposition

(5326)

2+ 1-

Related Posts:

1. كيف تُستخدم ال **UML** بشكل صحيح ؟

uml composition aggregation

تعليقات 5

RUAA بتاريخ 29 سبتمبر,2013 - 8:22 م

الرد

جزاك الله خير استاذ وجدي على المعلومات القيمة

متابعين باقي المعلومات عن هندسة البرمجيات ان شاء الله

بالتوفيق

saffor بتاريخ 18 يناير,2016 - 4:16 م

الرد

بارك الله فيك على المعلومات القيمة

الحبيب أمزال بتاريخ 28 يوليو,2016 - 7:21 م

الرد

بارك الله فيك.

تسلم اخي العزيز بارك الله فيك وجزاك الله خير كثيرا عن هذا المقال الرائع ...

ياسمين بتاريخ 9 مارس,2018 - 9:33 ص

شكرا جزيلا ..زادك الله علما ونفعا

الرد

أرسل

لن يتم نشر عنوان بريدك الإلكتروني. الحقول الإلزامية مشار إليها بـ *

التعليق

الاسم *

البريد الإلكتروني *

الموقع الإلكتروني

أرسل