تكلمنا سابقاً:

ضمن الـ structural : composite و decorator ومسألة مركبة عن singleton و decorator .

وسوف ننهي في هذه المحاضرة بـ structural و Behavioral ونتكلم عن الجودة وعلاقتها بالـ Design Pattern ، نأخذ مثال عن pattern

Structural Pattern

Adapter:

المشكلة: عندما يكون لدينا نظامين ، نظام خاص بي ونظام آخر أحتاج لربطهم.

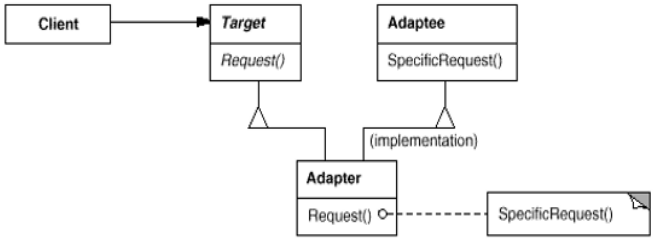
فلدينا حلين ، إما نقوم بتغيير النظام الآخر (interface) الخاصة به لجعله يتوافق مع نظامي ، أو أن نقوم بتغيير النظام الخاص بنا ، أو نأخذ الحل الثالث بخلق نظام وسيط بينهما وهو الحل الذي يحافظ على التصميم.

الـ Adapter هو object pattern و class pattern.

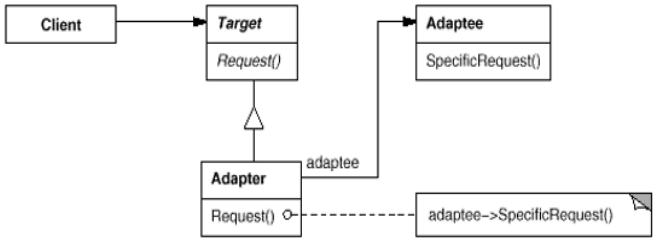
التعليل:

- هو class لأنه في حال كان لدينا نظامين ونريد القيام بـ adaptation بين target الذي أستخدم إمكانياته والـ adaptee ، فأنا أرث من الاثنين والـ request الخاصة بي أجعلها تنادي الـ specifiecReq من الـ adaptee .
- هو object: إذا الـ Adapter يريد الوراثة من الـ target والـ request الخاصة به تعمل composition لـ adaptee (أعرف كائن من adaptee ونقوم باستدعاء specifiecReq() الموجودة ضمن adaptee).

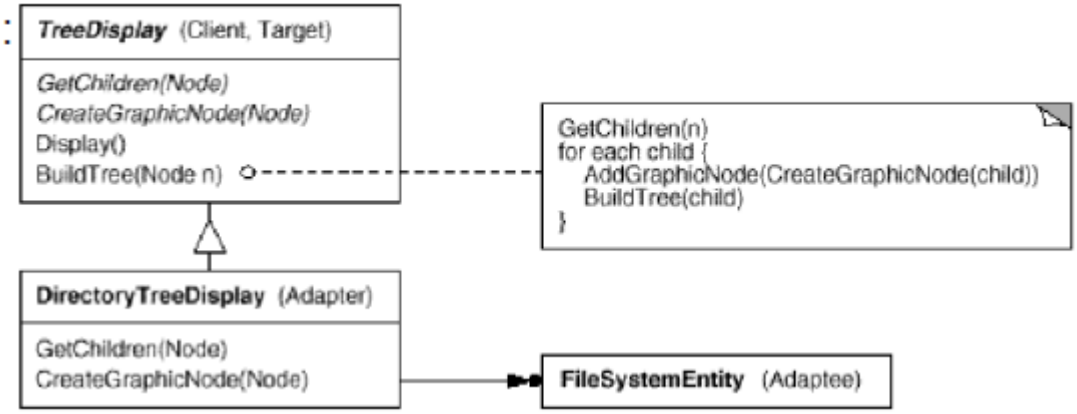
- Class Adapter pattern



- Object Adapter pattern



مثال: مكتبة لرسم الأشجار



لدينا بنية هرمية لرسم الأشجار وأحتاج هذه البنية لرسم شجرة الـ XML الخاصة بالملفات خلقت لتكون reusable.

فيكون أساس بنائها

Get children(node)

Create graphic node (node)

Display

يتم تنفيذهم بشكل عودي إلى أن يتم رسم الشجرة.

نريد استخدام File system entity وهي adaptee ويكون الاستخدام بعمل Adapter نسميه مثلاً DirectoryTreeDisplay ونقوم بعمل :

Override Getchildren(Node)

{

نقوم هنا ببناء node من خلال (file system) مكونات النظام فيما يتوافق معه (مكتبة reusable) ممكن الاستفادة منها في عملية إعادة بناء الـ node.

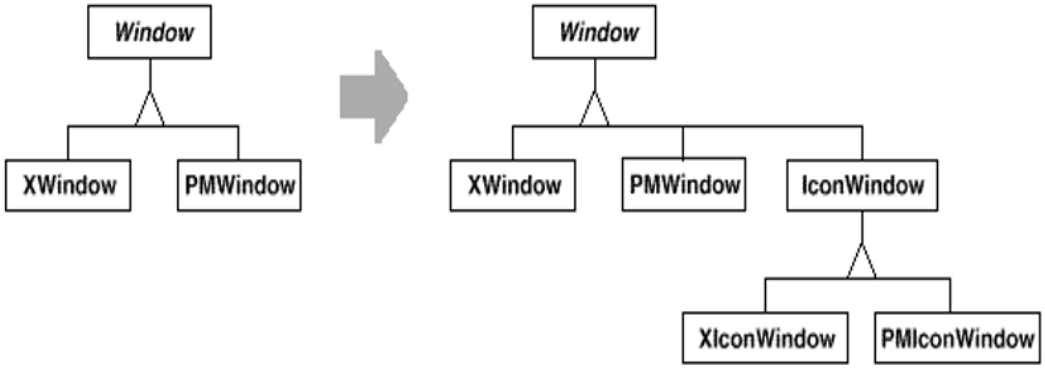
}

وبالتالي لم يتم تشويه الكود الأصلي وإنما يتم الوراثة ومن ثم صياغته فيما يناسب النظام.

نمط عملية التعديل على الـ method التي لا تناسب النظام يجب عمل وراثة ومن ثم override للـ method التي لا توافق النظام.

Bridge:

مشكلته: لدينا بنية شجرية (هرمية) مثل window ، لدينا أنواع منها Xwindow وPMwindow وبلحظة معينة يأتي نوع جديد (مفهوم) وهو iconwindow وهو مفهوم منطقي ونريد توزيعه للنوعين فتصبح الهرمية بالشكل التالي:



وبهذه الحالة تم الخلط بين الشيء المنطقي والذي هو المفهوم (نافذة بلا icon ونافذة مع icon) والشيء المادي والذي هو العائلات.

ويكون الحل:

ضمن الـ Bridge هو يفصل بين تجريد التنفيذ وتجريد التجريد.

ضمن class window لدينا ميثودين أساسيين :

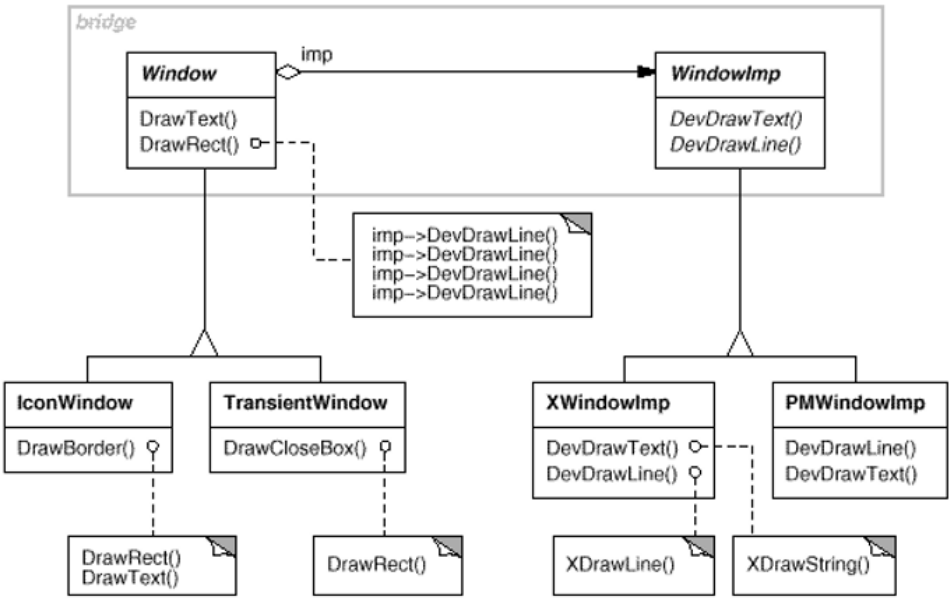
- Draw Text();
- Draw Rect();

نقوم بعمل تجريد لهم:

- Draw Text();
- Draw Line();

وأنا أعلم أن Xwindow سيكون لديها draw line,draw text ، ونفس الأمر عند PMwindow طبعاً مع اختلاف في طريقة البناء.

ولذلك يبقى التنفيذ (المخطط الأول) الذي يبين التنفيذ أي فيزيائياً نحن نلاحظ وجود Xwindow, PMwindow .



الفكرة باختصار لدي

الجسر يصل بين مفهوم التنفيذ والمفهوم المنطقي (التجريد).

مثال:

لدينا TableView وهي طريقة عرض على JTable ولدينا صف آخر وهو List view وتعتمد على JList.

وكلاهما ينفذان addLine وبالتالي توضع ضمن صف أب وهو BridgeImp .

الجانب المنطقي لدينا قائمة وقائمة مفروزة كلاهما يستخدمان addData وبالتالي يتم وضعها ضمن Bridge (صف أب) ولكن ضمن القائمة المفروزة نقوم بعمل إعادة بناء للـ addData لتقوم بالفرز ومن ثم يتم تنفيذ الـ addData عند الأب.

وبالتالي تم فصل ما هو متعلق بالبناء وما هو متعلق بالمفاهيم.

هكذا يمكن إدخال التنفيذ بالمفاهيم مع الحفاظ على هرمية صحيحة. وعند الإضافة يتم إضافة دون التعديل واستخدام التنفيذ بسهولة.

ملاحظة: حالات ورود حالات مثل Bridge ليست كبيرة بينما composite مع decorator هي كبيرة ، أيضاً حالات ورود Factory Method كبيرة. (هنالك نسب استخدام يجب الاطلاع عليها). والـ Adapter أيضاً يستخدم كثيراً لكن دون الانتباه لاستخدامه (كونه منطقي).

Proxy

تتوضح فكرته في مادة النظم الموزعة وعند شرح middleware.

مثال: Internet explorer قديماً كان بطيء لأنه لا يمكنه تحميل الصفحة التي تحوي الصور حتى تصل جميع المحتويات وبالتالي الصفحة التي تحتوي صور تأخذ وقت كبير.

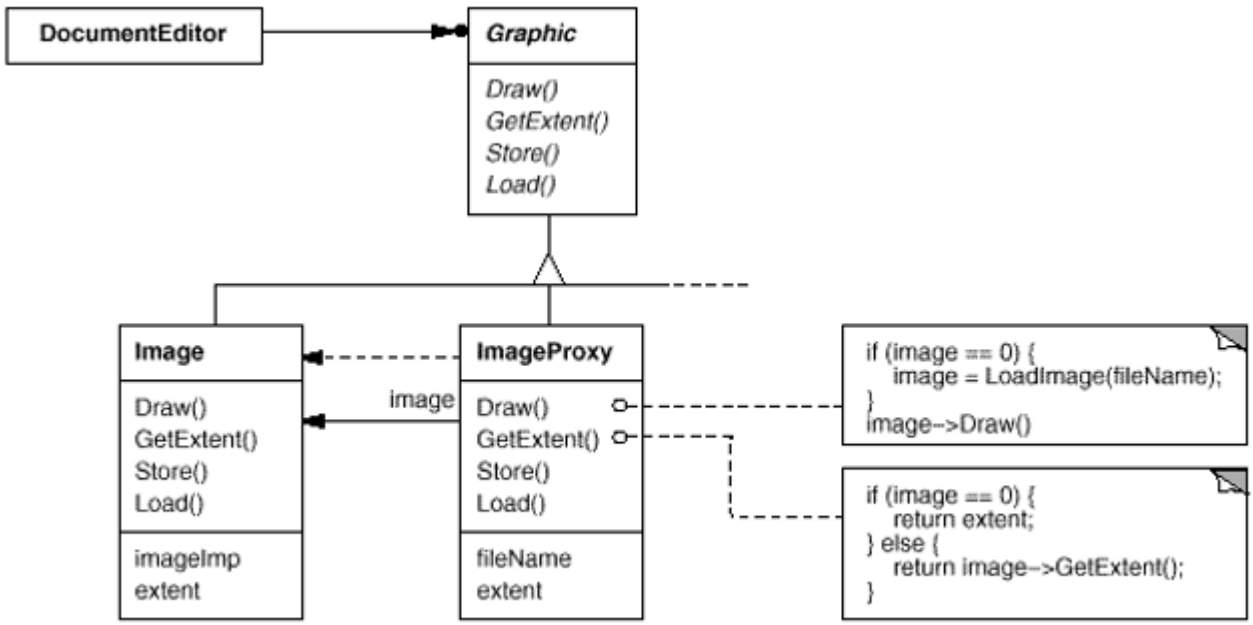
ثاني تطوير كان بالعرض الطبيعي للصفحة ولكن عند وصول صورة نضعها بمكانها ، مثلاً أنت تقرأ النص فتصل الصورة فيصبح النص مخربط.

والجيل الثالث من المتصفحات (ليس قديم جداً) عند تحميل صفحة وفيها صور والانترنت بطيء يعطي إطار للصورة والنص يُكتب طبيعي وعند وصول الصورة تتحمّل في مكانها. ==> هنا استخدم الـ Proxy.

على الانترنت تم تطوير هذه النقطة: ضمن الـ webserver عند إرسال صفحة مع صور ، يرى الصور ويرسل عوضاً عنها أغراض Proxy (بديل) يمكنك معرفة معلومات بسيطة عن الصورة (height , width) وبالتالي المتصفح يحجز مكان خاص بها.

((كائن object ينوب عن object إلى حد معين)).

مثال:



استخدامه:

1. **Remote proxy**

ضمن middleware (webserver ضمنها) وهو ليس غرضي التوجه.

أول برمجية ظهرت عام 1989 قبل ذلك عند كتابة تطبيق شبكي نستخدم TCP/UDP وكان يأخذ وقت طويل لأنه يتطلب فحص كبير. وكان موجود الفكر الغرضي التوجه (استقر عام 1970).

Object middleware: الهدف الأساسي الخاص به أن نستطيع برمجة تطبيقات شبكية مثل برمجة الغرضي التوجه المحلي (عدم تخيل أن الغرض بعيد).

مثال Stub & Skelton:

لدينا صف example_client داخله التنفيذ main نعرف كائن من نوع example ومن ثم نستدعي

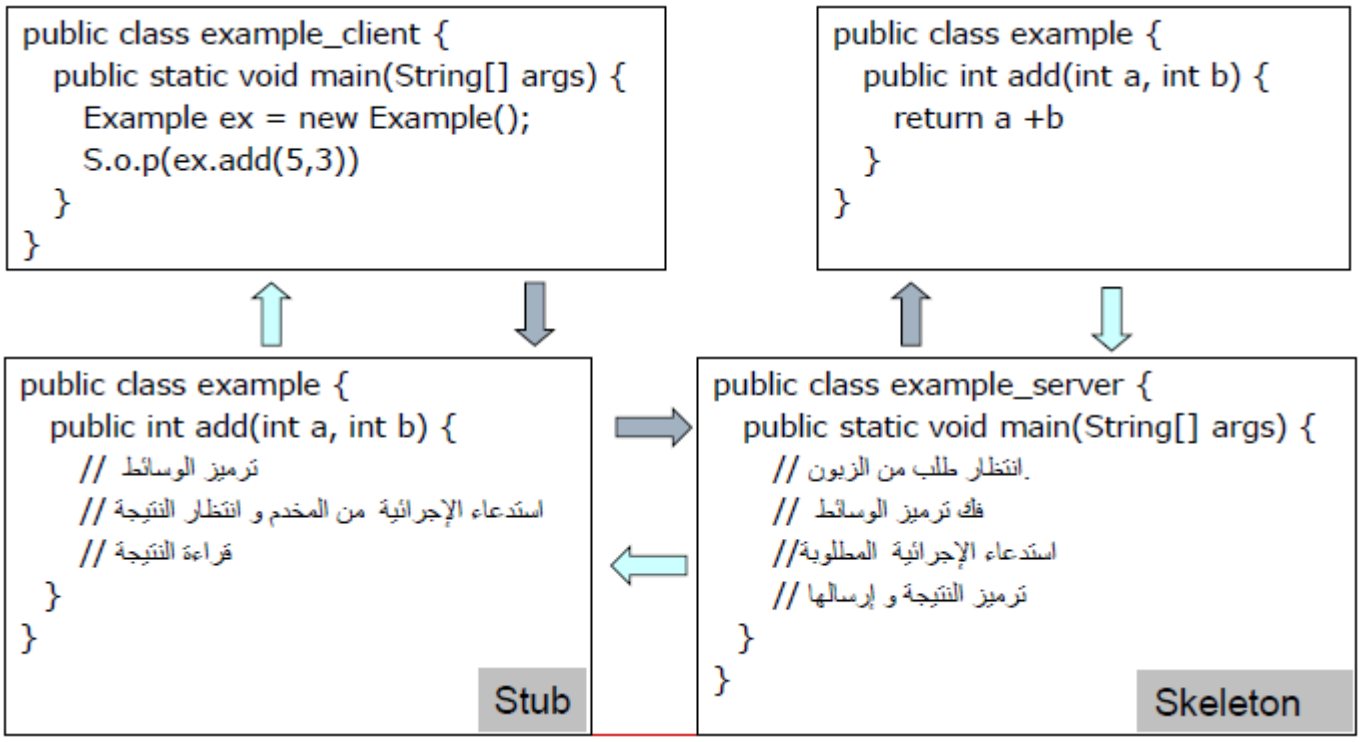
system of print(ex.add) ، أي تم استدعاء add وليتم ترجمته يجب أن يكون لدينا على نفس الجهاز صف اسمه example يحوي add ولكنها ضمن التنفيذ الخاص بها يتم ترميز الوسائط وأرسال الوسائط إلى المخدم وقراءة النتيجة من المخدم (ليس عملية الجمع الأساسية) وهو يتعامل مع الـ Skelton .

((تذكروا وظيفة الأمن)) وهو الـ server يحوي صف example_server ، ضمن الـ main يقوم بانتظار طلب الزبون (listener) ثم فك ترميز الوسائط التي جاءت إليه TCP/UDP ثم استدعاء الإجرائية add ضمن الصف الحقيقي example الذي يحوي العملية الحقيقية ويمكنه استدعاءه لأنهما على نفس الحاسب وبعد الاستدعاء يقوم بترميز النتيجة وإرسالها.

في النهاية Stub وSkelton غير ظاهرين للمبرمج فعلياً.

رأى المبرمج أن لدينا إجرائية add ضمن الصف example ضمن جهاز ثاني وأنا لدي reference منه وأقوم باستخدامه ولكن هذا الـ ref هو عبارة عن Proxy.

وبالتالي middleware تقوم بتسهيل البرمجة ولا تدخل في التفاصيل وإنما تكون الأمور مجردة.



فائدة الـ remote proxy: إخفاء تفاصيل الشبكة.

2. **Virtual Proxy:**

أحياناً يكون لدينا أغراض كبيرة جداً بدلاً من تحميلها في الذاكرة نقوم بتحميل بديل عنها بسيط يمكنه رد العمليات الأساسيّة ، في حال تم طلب عملية كبيرة نقوم بتحميل الأصلي من الذاكرة (تذكر الـ virtual memory التي تم شرحها في مادة البنيان).

3. **Protection Proxy:**

مثلاً لدينا غرض وله اتصال بالـ database (ونحن نعلم أن الناس يمكنها الوصول للأغراض عن طريق الشبكة) وبالتالي لعمل درجة حماية أكبر أقوم بعمل Proxy. لحمايته وعند حدوث خرق يتم الوصول لـ Proxy ولا يمكنه الوصول لـ database. وبالتالي تم الفصل بطبقات.

4. **Smart reference**:

يتم شرحها وفهمها بشكل واضح في مادة النظم الموزعة.

عبارة عن غرض موجود بكل بيئة يقوم بعدّ الأغراض الـ Proxy والـ garbage collector ضمن خوارزميته يرى أنه في حال كان هنالك remote ref ولا أحد يستخدمه يحذف ، ولكن هنالك مشكلة أنه في حال حذف ref قام بإرسال رسالة فعند وصولها يعطي خطأ null reference نقوم بحلها عن طريق الـ snapshot لحالة النظام لكل المواقع الموزعة ونضيف عليه أنها تأخذ حالة القنوات ، إذا كان هنالك remote ref على الشبكة نستطيع الوصول إليه وبالتالي لا يمكن حذف smart ref إذا كان يقوم بعملية.

ملاحظة : proxy هو الذي خلق الـ middleware ويتم استخدام هذا الـ design pattern ضمنه بشكل كبير.

- **Flyweight**

على فرض نريد القيام بـ object editor text يعني كل حرف يمثل بـ object وفيه مواصفات معينة (صورة ، لون ..) فإذا كان لدينا نص من مئات الأسطر أي من مئات الآلاف من المحارف وبالتالي لدينا عدد هائل من الكائنات داخل الذاكرة ، وبالتالي يجب أن تقوم بفصل الـ object ضمن البرنامج من ناحيتين ، الأولى المعلومات الثابتة والثانية المعلومات المتغيرة.

هذه الأحرف مشتركة بعدة واصفات (مثال شكل حرف a نفسه لا يتغير) فمن غير المعقول أن نقوم بإنشاء كائن عند المرور بكل كلمة تحتوي على الحرف a ، وبالتالي الحل الأفضل أن نقوم بإنشاء كائن واحد في الذاكرة ونولد (reference) مرجع خاص عند المرور بكل حرف a.

ولكن هنالك معلومات متعلقة بالسياق (رقم السطر ، رقم العمود ، ...) ومعلومات ثابتة مثل صورة الحرف (هي التي تشغل الحجم الكبير).

ملاحظة: الخط المائل هو صورة عن الحرف ، بحيث السياق سيقوم بتحديد أيّ صورة يحتاجها لرسم الحرف.

وبالتالي نقوم بعمل pull من object وعند كتابة كلمة ألقي نظرة على pull في حال لم أجد الحرف الجديد المكتوب ضمن الكلمة السابقة أقوم بإنشاءه (يكون من نوع singliton factory) ونقوم بعمل ref له ضمن الكلمة وأضيف معلومات السياق(رقم السطر..).

نتيجة : يتألف هذا التصميم من قسمين:

1. Concrete flyweight وتحتوي الصفات المشتركة.
2. Unsaved concrete flyweight تحتوي معلومات السياق.

(راجع السلايد 51 من structural patterns)

استخدامها:

مسائل امتحان: مثال لدينا برنامج للآليات في سورية فإن مواصفات السيارات معروفة ، مثلاً سيارة Lada معروفة المواصفات (كم مقعد تمتلك ، استطاعة المحرك ، السعة..) وعلى فرض لدينا مئة ألف سيارة ؟! هنا أنشئ غرض واحد ونقوم بعمل ref يحوي اسم المالك ، اللون ، الرقم..

- **Façade**

وهو مصطلح فرنسي وتعني واجهة ، ويتجلّى دوره بعمل واجهة لمخاطبة العالم الخارجي وبالتالي نقوم بتوليد صف يستطيع التواصل مع كل البنى الداخلية للنظام ، ومن ثم يقوم هذا الصف بالتواصل مع العالم الخارجي.

وبالتالي العالم الخارجي لا يستطيعون معرفة أسماء الـ objects الداخلية بالنظام وهذا يعطي درجة أمن وسهولة الاستخدام وبالتالي جميع الطلبات تأتي لهذا الصف ويمكن عمل replication لتجنب الحمل الزائد على الغرض يمكن عمل غرض ثاني على نفس المستوى بنفس الخدمات.

ويتم العودة لمفهوم الـ replication ضمن مقرر النظم الموزعة.

الفرق بين proxy و facade :

الـ proxy يقوم بعمل واجهته ولكن من غرض واحد ، أما الـ facade فيكون لكافة كائنات النظام. (دائماً الـ client يعرف الـ server ولكن server من غير الضروري أن يعرف الـ client).

Behavioral patterns

وهي تحل مشاكل الخوارزميات والسلوك بين الأغراض.

1. Template method

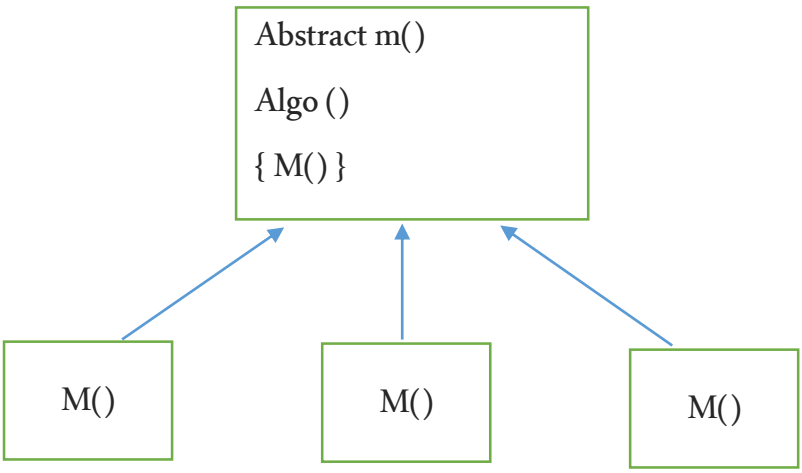
بسيط جداً ومستخدم بكثرة ، يستخدم عندما يكون لدينا خوارزمية مؤلفة من مجموعة من الخطوات تنفيذها يختلف حسب الـ subclass ، كونها خوارزمية عامة لا داعي لإعادة كتابتها عند كل صنف من الأصناف وإنما نقوم ببنائها وكتابتها في مكان واحد.

الخطوة التي تكون متغيرة نقوم ببنائها على أساس abstract method و subclasses نقوم بإعادة كتابتها بالطريقة المناسبة لها.

مثال: أساتذة الجامعة أعطى طريقة لحساب الساعات الإضافية الخاصة بهم.

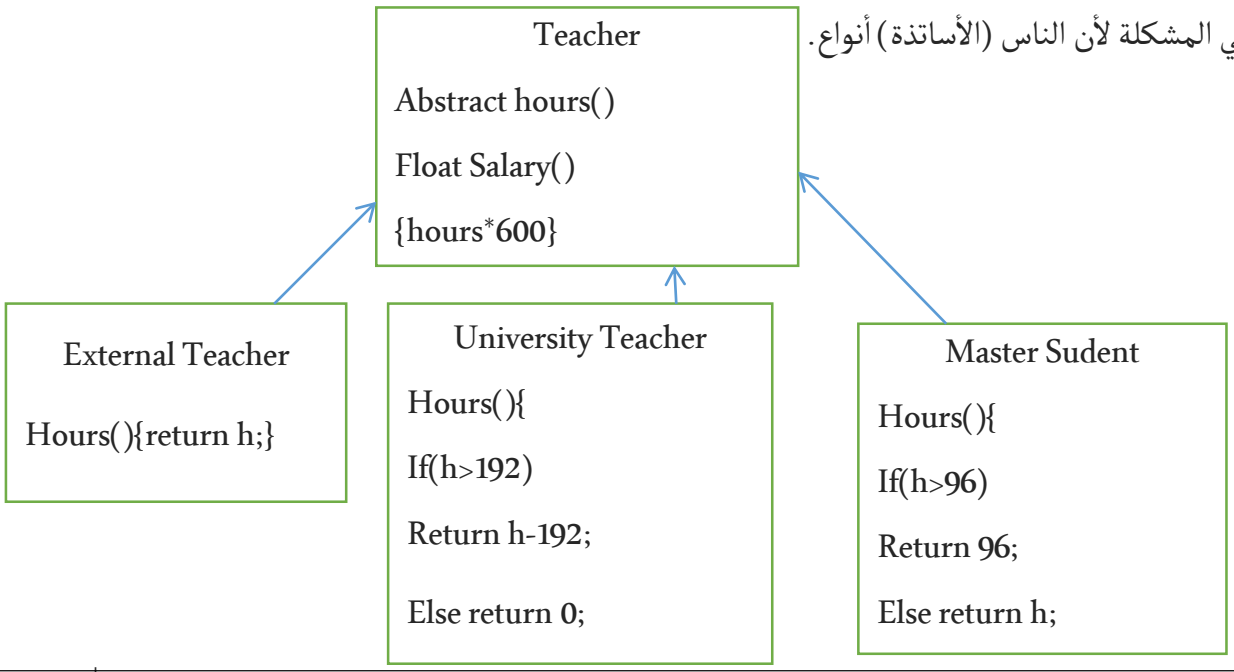
لدي ثلاث أنواع من الأساتذة :

- extend teacher : كل ساعات تدريسهم تعتبر ساعات إضافية.
- university teacher (موظف في الجامعة) : ساعاته الإضافية هي التي تتجاوز 192 ضمن الفصل.
- master student : طلاب دراسات عليا كل ساعاتهم إضافية بحيث لا تتجاوز 96 ساعة ، له حد وذلك كي لا يصرف وقته في التدريس.



وخوارزميتنا تكون بحيث تضرب 600 بعدد الساعات.

حساب الساعات هي المشكلة لأن الناس (الأساتذة) أنواع.



مثال: مسألة نقل البضائع

نقوم بعمل حاويات لنقل البضائع ، كل بضاعة لها حجم ووزن وعند عمل حاوية container نضع فيها مواد حسب نوع النقل ، إذا كان بري نركز على الوزن ، وإذا كان بحري أيضاً نركز على الوزن ، أما في حال كان جوي فهو يركز على الحجم.

ولدينا جدول يوضح الكمية والكلفة وفقاً لنوع النقل (سلايد 24 behavioral).

فمثلاً النقل البحري تستطيع وضع كمية تصل لـ 30 ألف كغ ونركز على الوزن ...الخ

ملاحظة: هذه المسألة هي مسألة نموذجية للتدرب على القيام بعمل برامج غرضية التوجه ، برامج غرضية التوجه خلقت لتجميع المفاهيم المشتركة (عامة) common concept. وهذا هدف الـ template.

هذه هي فلسفة البرمجة غرضية التوجه ، طالما نستطيع أن نقوم بعملية تجميع فنجمع ونستخدم abstract method لنعطي التنفيذ للصفوف الأبناء (لا نعيد كتابة شيء).

2. **Observer / subject**

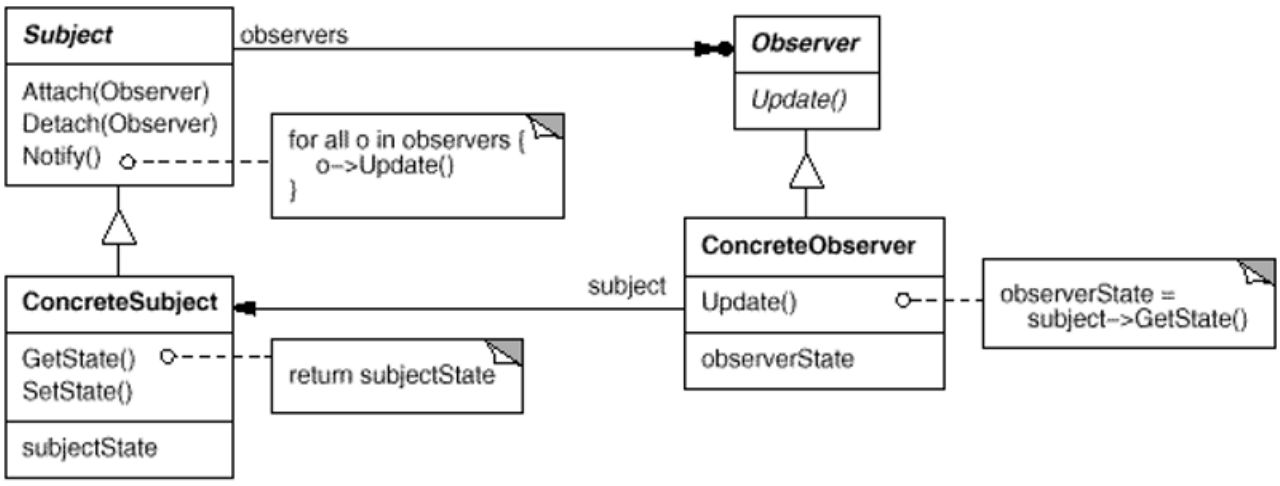
وهو من أهم الـ design pattern ومستخدم بكثرة شديدة ، كل الأنظمة التي تتعامل مع events هي تستخدم observer.

مثال (UI واجهات عند ضغط زر مثلاً ، حساسات بمبنى ، smart home..)

قاعدتنا نحن لا نعيد تصنيع الدولاب وإنما نقوم بتحسينه.

الفكرة: لدينا غرض اسمه subject يحوي data ولدينا أغراض كثيرة تراقبه (observer) فإني أحتاج آلية لنقل البيانات للآخرين عند تغييرها.

يمكن تطويره أي في حال تغير ضمن الـ observer مثل (مثلث) قم بتكبير المثلث بالفأرة فالمفروض كل الأعمدة أن تكبر وكل الجداول أن تتغير.



شرح توضيحي:

على فرض لدينا دائرة ومربع ... ونريد تكبير الشكل ، بداية أن الدائرة وكل الأشكال تعتبر concrete observer فعند القيام بحدث التكبير مثلاً فإن الصف الخاص به يتصل مع concrete subject ومنه يأخذ المقدار الذي تم عليه التعديل.

وممكن أن تكون حالة إزاحة شكل فنستخدم set state تأخذ ثلاث قيم A,B,C يتم تثبيتها لديها ويعمل notify ويتم عمل update وبالتالي كل الأشكال تعدل قيمها وفقاً للتابع update الخاص بها والتي تحوي مثلاً get state التي تأخذ قيم A,B,C وتقوم بحفظها ، ويمكن التطوير بإضافة set state لتعديل A,B,C ومن ثم تقوم بعمل notify للتعديل لدى الكل ليكون نظام متوافق (هام جداً).

ضمن الجافا لدينا الـ source event والتي هي subject وجميعهم يضيفون listener ويخدموا listener (مستمع) و notify.

تم التوضيح ضمن المخطط الزمني (سلايد 28) ، بلحظة من اللحظات concrete observer يعمل set state ينادي notify وهي تذهب وتستدعي update يعمل get state ومن ثم يتحرك لـ concrete observer الثاني ويعمل update .

الغرض : جاء هذا التصميم لحل مشكلة أنه يوجد غرض actor لا يجب أن يعمل عندما يلاحظ أن حدث آخر يعمل ، لكن ليس بالضرورة أن يعي هذا الأخير بوجود actor.

مثال: الدكتور يلقي محاضرة مهما كان العدد وهو يمثل subject والطلاب reactor observer يمكنهم أن يكونوا من طبيعة مختلفة وأن يتغيروا دون أن يشعر بذلك المصدر.

مثال: إدارة الأحداث ضمن الجافا awt/swing

مثال: في السلايد 30

يرن الهاتف فإن عدداً من الأشخاص أو الآلات يمكن أن يكونوا معنيين بالاتصال إن الأشخاص يمكنهم أن يدخلوا ويخرجوا من الغرفة ، وبالتالي الدخول والخروج يحدد الانتفاع من الهاتف.

مراحل التصميم ضمن الجافا:

جافا قدمت نموذج الأحداث وعملت template كبيرة وواضحة ماهي صفوف الأحداث التي ترث منها لتقوم بعمل حدث جديد ولديك listener لتقوم بعمل observer وتعرف الصف المصدر للأحداث. (السلايد 31).

1. تعريف صفوف الأحداث: (java.util.EventObject) الخاص بالأحداث وتقوم باستدعاء ضمن الباني ;super(source) لترسله للآخرين في الجافا عند خلقه observer listener هي تتولى عملية إرسال الحالة والتي هي مصدر الحدث.
2. واجهات listeners: للتعريف java.util.Eventlistener تعرف داخله العمليات التي تحتاجها ()telephoneRange و ()telephoneAnswer عملية الرن أو الرد هي الأحداث التي يمكن أن تتحسس لها وأنت في الغرفة.

وتعريف صفوف Adapter وهي خيارية.

بدل من وراثة من listener محدد نقوم بالوراثة من Adapter له.

الفائدة: مثلاً ضمن الصف الأساسي listener لديك 10 method ولكنك تعمل ضمن واحدة فيتم وضعها ضمن Adapter ونتعامل معه ، سبب كثرة العمليات ضمن الـ listener لأنه يرث من abstract class فيجب أن يحوي جميع العمليات.

(سلايد 35)

نقوم بتعريف قائمة من Telephone Listener مع Add و Remove للإضافة والحذف من القائمة fireTelephoneRang نقوم بالدوران ضمن القائمة ويعطيه حدث telephone rang ويعطيه event .(سلايد 36)

مثال: في حال كان لدينا Answer machine عند الاتصال تقوم بطباعة رد وقد تم عمل class adapter لاستدعاء Rang فقط ، و person يستخدم الـ adapter للرد على الهاتف.

3. Chain of responsibility

مثال: ضمن الجافا هو exception نظام الاستثناءات هو نظام تسامح مع الأخطاء إذا تم ارتكاب خطأ لن تكون نهاية البرنامج الخاص بك ، إنما هناك مجال للإصلاح.

لدينا حالتين لإلقاء الاستثناء throw ويمكنني استخدام throws للسماح له بالمرور .

لماذا استخدم الحالتين ، استخدم throws لأنني لا يمكنني أن أعالج (ليس لدي المعرفة لأعالج) ، مثل class overflow هو لا يعرف ماذا سيفعل في حال هذا الصف ولكن من يقوم باستخدامه ، لاحظ عند الإضافة الـ overflow وبالتالي يقوم بخلق stack ثاني ويضيف عناصره.

Stack مهمتي فقط وضع عناصر فيه وأعيدها عند الطلب والذي يلاحظ الـ overflow هو من يستخدم هذه الـ stack.

Throw(exception) تعني لدي مشكلة وغير قادر على حلها.

معلومة: عند إنشاء الـ java كانت design pattern مستقرة تماماً وتم تطبيق جميع مفاهيمها ضمن هذه اللغة.

Design pattern and software quality

كيف سيتم قياس تأثير الـ design pattern على الجودة الخاصة بالبرمجيات .

مقدمة :

هل التصاميم السابقة تؤثر على جودة البرمجيات ؟

<< بالبحث العلمي لا يمكن أن تقول شيء إن لم تستند على تقارير وعلى أسس >>

على ماذا استندوا في أجوبتهم على هذا السؤال:

1. الدراسة التجريبية: ضمن هندسة البرمجيات تكون الدراسة التجريبية نادرة ، لسنا في علوم إنسانية أو اجتماعية ، نظم المعلومات نظام غير ملموس. تم الاستعانة بأربع خبراء بالـ design pattern يهتمون بالخصائص ومنها (understandability,reliability,reusability,extandability) فيتم إعطاء كل خبير تصميم معين ويقوم بتقييمه generality إما بالإيجاب أو النفي ضمن هذه الخصائص وبعدها يتم التحليل وتظهر النتائج مثال composite ليس له آثار سلبية.

حالياً يتم القيام ببحث ضمن MDE (تخلق process جديدة) لكن مفرداتها ليست class, association وإنما مفرداتها العمل والواقع --> خط إنتاج ، فتظهر meta model كبير ويمثل مفردات لغة النمذجة مثل لدينا meta class,meta association عند خلق صف أو علاقة تكون قد خلقت object منها ، فعند تحليل النظام يظهر meta model كبير فنحن سنقوم بتقسيمهم ونحن سنولد أداة نمذجة ، فأفضل طريقة لتقسيم الـ meta model بالتجريب نضع meta model ضمن أدوات النمذجة بحيث إذا تم التقسيم لـ meta model واحد سيتم خلق خط إنتاج وحيد وهذا لا نريده(قسمين خطين إنتاج..).

وبعدها يتم قياس مدى الفعاليّة ، من ثم تم طلب بناء model معتمد على السلسلة المبنية وقم بعد عدد النقرات للوصول لها -- > وهذا لتحديد reusability وأعطي تقارير بالنتائج.

هنالك ملفات help تأتي مع النمذجة وطُلِب من الطلاب قراءتها وذلك لتحديد مدى فعالية الفهم.

وبالتالي نأخذ هذه المعلومات لصنع خوارزمية تفصل الـ meta model لـ meta model أصغر للحصول على سلسلة أدوات تكون reusability.

وهذا كان عمل لسنوات سابقة.

أما خلال هذه السنة فإن السيمنار سيكون ضمن Maintainability quality يأتي منتج (نظام) لدينا عدة واصفات ساكنة مثل عمق شجرة الوراثة ، تعقيد الميثودات (الوظائف) حجم الكود ، حجم المتحولات.

هل هنالك رابط بينهم ؟! سيتم حالياً إدخالهم على أدوات data mining (عدة أمثلة) لنعرف ترابط الأمور من التجريب.

مثال هل هذا النظام سهل التعديل ؟! يتم إعطاءك وظيفة معينة لإضافتها فسيتم قياس كم دقيقة استغرق التعديل... ويتم وضع نتائج في تقارير ، وبالتالي يتم معرفة درجة الجودة ويكون الخرج نظام أعطيه مشروع يقوم بقراءة model ويعطي تقييمه على خصائص الجودة السابقة.

انتهت المحاضرة

Written by:

Hamida Abo Rshed

Word press and preparation:

Enas Alhalabi

Reviewed by:

Eman Zyadeh