Object Oriented Design, Design Pattern & Implementation Issues

التجريد (Abstraction) :

أحد المفاهيم الأساسية في البرمجة غرضية التوجه، هدفه الرئيسي التعامل مع التعقيد وتبسيط التعامل مع النظام، نبدأ باستخراج الأغراض الرئيسية ولمشكلة للنظام وأسماءها ثم نبحث عن الattributes التي نحتاجها والmethods لكل غرض والخدمات والمسؤوليات التي يقدمها الغرض حيث وجود الغرض (object) يعني وجود خدمة يقدمها عن طريق الmethods التي يحتويها.

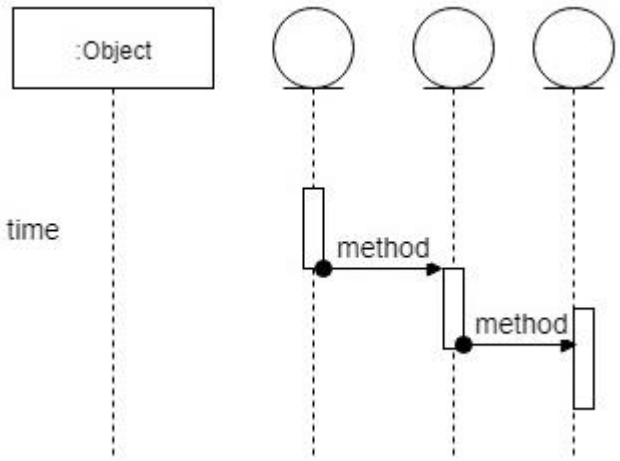
كل النظم غرضية التوجه تطور بشكل تكراري تزايدي :
مثال : نريد تطوير نظام تعليمي نبدأ بإنشاء 3 أغراض [course, student, teacher] وهي مرحلة التحليل ثم نأخذ أول غرض teacher ونفكر بال attributes وال methods الخاصة فيه عبر وسائل جمع المعلومات المختلفة وعن طريق التجريد نقوم بعزل المعلومات المهمة عن المعلومات غير الهامة ونستفيد من طريقة Class CRC Responsibility Collaboration ونطبق مفهوم Chasings، Cabling وعن طريق مراعاة هذين المفهومين يصبح لدينا فكرة عن الModularity.

Encapsulation

حزم الداتا مع التوابع الخاصة بها عن طريق الكلاس والحزم (package) مما يؤمن تغليف للداتا.
إن التفكير بالمنهجية غرضية التوجه يعني تقسيم النظام إلى أنظمة جزئية (كلاسات) ثم نعرف الأغراض من تلك الصفوف وأخيراً تعريف الmethods الخاصة بكل كلاس وما تقدمه من خدمات.

Sequence Diagram

مخطط ثنائي الأبعاد، المحور الأفقي يمثل محور الأغراض أو الأنظمة الجزئية وكيف تتفاعل مع بعضها والمحور العمودي يمثل الزمن



نبدأ بهذا المخطط في مرحلة التصميم فهو مخطط تصميمي :

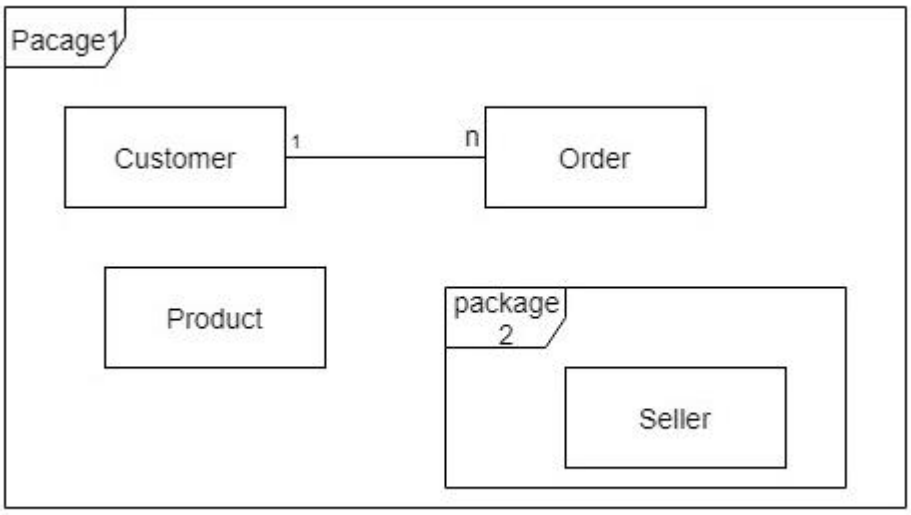
Collaboration Diagram

يركز على التفاعل بين الأغراض عن طريق الرسائل (messages) .

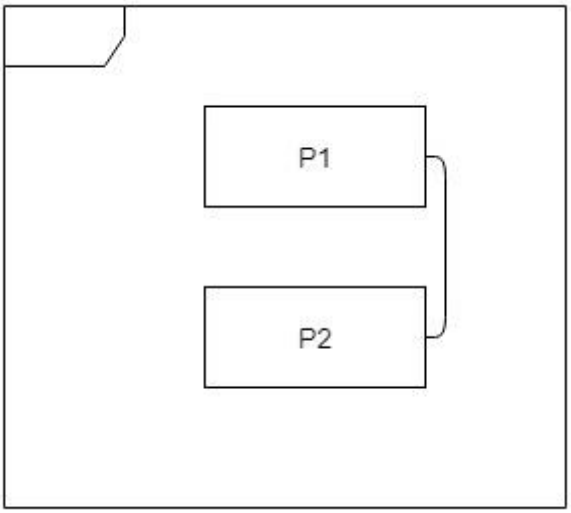
Package Diagram

Package : آلية لحزم مجموعة من الصفوف وال **interfaces** والحزم الداخلية المرتبطة مع بعضها بهدف زيادة التغليف وإعادة الاستخدام.

مخطط الحزم مخطط أساسي وبفرض لدينا مخطط الصفوف التالي :



يكون مخطط الحزم الموافق :



وهو مخطط معماري يعبر عن الأنظمة الجزئية وكيف تتواصل مع بعضها، يتم استنتاجه من مخطط الصفوف، غالباً أي نظام يتم تحويله إلى 3 حزم واحدة للداتا وواحدة لل presentation وواحدة للتواصل بينهما.

قبل الانتقال للتصميم غرضي التوجه يجب أن نقوم بتحليل غرضي التوجه حيث في التحليل نتعرف فقط على أسماء الصفوف وبعض الattributes أما الMethods نحددها في مرحلة التصميم ثم تأتي مرحلة التكويد لتحويل مخطط التصميم إلى كود قابل للتنفيذ.

في التصميم يجب التركيز على الQuality (الجودة) :

التصميم يتأثر بالمتطلبات غير الوظيفية من خلال coupling و cohesion.

ملاحظة: المتطلبات غير الوظيفية تقسم لثلاث أقسام :

متطلبات متعلقة بالمنتج مثل الأمان والأداء وهي متطلبات جودة، متطلبات خارجية (قيود) متطلبات أخلاقية (قوانين).

صفات التصميم الجيد :

1. يجب أن يحقق المتطلبات الواضحة (المتطلبات الوظيفية) والمتطلبات الضمنية (الغامضة) وهي المتطلبات غير الوظيفية.
2. يجب أن يكون التصميم واضح وسهل الفهم والقراءة لذلك يجب استخدام المخططات والطرق المعيارية (مثل SRS).
3. أن يكون سهل الاختبار : في اختبار التكامل نحتاج العودة إلى مخططات التصميمية (معمارية النظام) لمعرفة الأنظمة الجزئية التي تتواصل مع بعضها.
4. يعطي صورة واضحة حول النظام للاستفادة منه في التكويد ومراعاة القيود الفيزيائية والبيئة الشبكية.

Generic Design Process

نبدأ من النموذج التحليلي للوصول للنموذج التصميمي بالاعتماد على وثيقة SRS حيث المخططات التحليلية هي : Activity Diagram, Class Diagram, Use Case Diagram أما المخططات التصميمية هي Sequence Diagram, Collaboration Diagram, State Diagram, Package Diagram

تحديات في التصميم غرضي التوجه :

1. أهم تحدي هو إيجاد الكلاس الحقيقي عن طريق الخبرة وطريقة الأسماء وUse Case فهناك أسماء واضح أنها صفوف وأسماء واضح أنها attributes ومجموعة أسماء غير محددة.
2. تعريف الواجهات الخاصة بكل نظام جزئي.
3. مراعاة إعادة الاستخدام أثناء التصميم وهي عن المتطلبات غير الوظيفية المتعلقة بالمطور.
4. مراعاة التجريد : كيفية التعامل مع المشكلة بالتجريد بداية من الصفوف ثم methods و attributes.
5. Information Hiding : من خلال Encapsulation عن طريق الclass والpackage.
6. High Cohesion, low Coupling : ودرجة ترابط العناصر ضمن الصف الواحد تكون كبيرة عندما تكون بحاجة لأداء عالٍ.
7. Modularity : تقسيم النظام لأنظمة جزئية اعتماداً على Cohesion, Coupling.

ملاحظة : الفرق بين مخطط الصفوف ومخطط الحزم:

- ✓ مخطط الصفوف يعطينا نظرة بنيوية عن النظام من خلال الصفوف الموجودة والعلاقات فيما بينها.
- ✓ مخطط الحزم : يعطينا نظرة حول الأجزاء الفعلية للنظام ويمثل مخطط تصميمي للنظام بشكل مجرد دون النظر للتفاصيل.

Message Design (Responsibilities)

وهي الmethods لا تظهر حتى مرحلة التصميم ويمكن أخذها عن sequence Diagram أو collaboration عندما يتواصل object مع آخر فهذا يعني استدعاء method موجود في الobject الثاني وجسم هذا الmethod هو اسم الرسالة بالإضافة للبرامترات الخاصة به.

معايير تصميمية

1. مفهوم التفكيك : درجة ترابط الأجزاء مع بعضها وبالتالي له علاقة بالModularity وال Coupling cohesion والمتطلبات غير الوظيفية.
2. التشكيلية : التفكير بإعادة الاستخدام ومحاولة استخدام knowledge Reuse.
3. الفهم : التصميم بشكل مفهوم من حيث أسماء مناسبة معبرة للمساعدة على فهم النظام.
4. تجميع الصفوف المتشابهة التي تتغير مع بعضها في حزمة واحدة لتكون قابلة للتعديل مستقبلاً.
5. الحماية : نعتمد على معمارية الطبقات والداتا التي نريد حمايتها تكون في الطبقات الداخلية وبالتالي الوصول لها يكون صعب.
6. استخلاص الواجهات Graphic User interface من خلال Human Interaction Component.
7. Problem domain Component : يقدم فهم للمشكلة ومنه فهم للصفوف التي نحتاجها لايجاد حل لتلك المشكلة وما هي الخدمات التي تتم على التوازي وإدارة المهام كيف تحدث (على التوازي ،على التسلسل).
8. Data management Component : لمعرفة شكل قاعدة البيانات وتصميم الdatabase وبالتالي كل جزء من هذه الأجزاء يعطينا فكرة عن التصميم.

ملاحظة : يوجد object oriented database وهي نوع خاص من الداتا بيز قليلة الاستخدام ومعظم النظم تستخدم الطريقة العلائقية(الجداول) وهي أبسط وأسهل للمصممين.

مجموعة من خطوات (best practice) :

1. Partition : كيفية تقسيم النظام.
2. الأنظمة التي تعمل مع بعضها في لحظة معينة.
3. User interface.
4. Task Management
5. Data Management.
6. Enter Subsystem communication : الأنظمة الداخلية كيف تتواصل مع بعضها.

هذه المبادئ لا يمكن مراعاتها جميعها في نفس الوقت فيجب وضع أولويات للمبادئ والأهم وأخذها بعين الاعتبار.

مثال عن تقسيم النظام : نظام Client-Server يمكن أن نعتبر الClient وهو Component ،والServer هو component ويمكن أن يتواصلوا عبر بروتوكلات لو كان web server نستخدم http protocol mail server التواصل عبر SMTP و File Server التواصل عبر FTP

وقد يكون التواصل على شكل **method Call** للتواصل بين **class** وآخر.

خرج مرحلة التصميم هو المخططات التصميمة كاملة حيث ننتهي من مخطط الصفوف ومخطط الحالات ومخطط الحزم.

ملاحظة: الفائدة من مخطط الحزم هو إعادة الاستخدام بدل أن نقوم بتطوير النظام كامل نستعين بحزم جاهزة تكون أوفر وأكثر جودة

مرحلة التكويد :

تحويل التصميم لكود بالاعتماد على المخططات مثل مخطط الصفوف باستخدام لغة برمجية والهدف منها جعل الكود **readable** (قابل للقراءة) تكون أسماء المتحولات والتوابع واضحة ومعبرة والخدمات التي يقدمها واضحة ويوجد تعليقات بهدف جعل التعديل عليه سهل من أجل إضافة الخدمات أو تصحيح الأخطاء.

Build or Buy

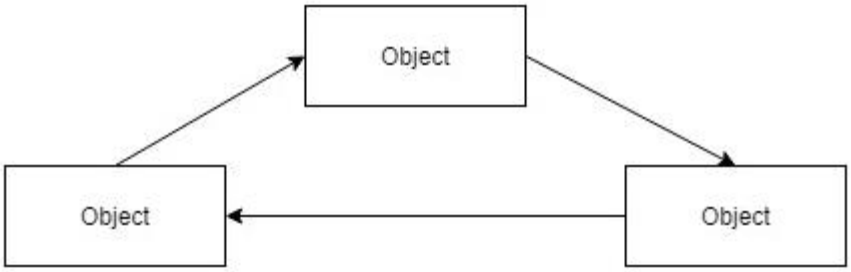
قبل البدء بعملية التكويد يوجد خيار يمكن أن نعتمد عليه وهو الاستفادة من الحزم الجاهزة لنقوم بشرائها ثم نضيف الخدمات الخاصة بنا وهذا ما يسمى إعادة الاستخدام وهو يزيد الجودة ويخفف الكلفة.

إشارات جيدة للتفكير غرضي التوجه:

- التوابع بسيطة وقصيرة وغير معقدة.
- عدد المتحولات المنشأة قليل.
- مهام الـ**object** واضحة، يمكن أن نضع الخدمات كلها في صف واحد لكن هذا سيئ فيجد تقسيم النظام لصفوف**some principles** .

1. **The Dependency Inversion Principle** الاعتماد على **interfaces** ألا نعتمد على **instance Class** مما يزيد الـ**reusability** فيصبح مجال استخدام الصف أوسع ويساعد في العمل الجماعي حيث يتفق فريق التطوير على الـ**interfaces** الأساسية ثم يتم تقسيم العمل فيما بينهم.
2. **The Interface Segregation Principle** لا نقدم واجهة واحدة للمستخدم بل نقدم عدة واجهات كل واجهة تقدم خدمة ما أو مجموعة خدمات.
3. **The Acyclic Dependencies Principle**

لدينا مجموعة الحزم



P_1 تعتمد على P_2 و P_2 تعتمد على P_3 هذا نموذج سيئ عند حدوث أي خطأ في P_1 ينتقل إلى P_3 و P_2 فنحاول الأبعاد عن الاعتمادية الدائرية وكسرها.

4. The Open-Closed Principle

هو مبدأ مهم تصميمي تكويدي الهدف منه جعل الصف قابل لإضافة الخدمات لكنه مغلق بالنسبة للتعديل عليه وهذا حيقدم حماية للنظام.

5. The Common Closure Principle : الخدمات المتشابهة تذهب مع بعضها في package واحد.
6. The Common Reuse Principle : الخدمات غير المتشابهة لا تذهب مع بعضها.

Refactoring

مبدأ تصميمي تكويدي لا يدل على الوظائف وإنما فقط الشكل بحذف المتحولات غير المستخدمة وتغيير الأسماء لتصبح أسماء المتحولات والتوابع معبرة مما يقدم حماية للنظام وعمر أطول وتجنب للأخطاء وصيانة مسبقة ويتم بشكل دوري ،كل IDE تحوي Automatic Refactoring.

The law of Demeter

1. Don't Talk to Strangers : لا ننشأ علاقة بين صفين لا يعتمدان على بعضها.
2. Don't Send Messages to Objects returned from other message sends.

Code smells

تجنب ما يلي :

- Duplicated Code : يتم حله عن طريق الوراثة.
- Long Methods : تقسيم التوابع الطويلة إلى توابع أبسط ليصبح أسهل للقراءة.
- Large Methods : نضع كل الخدمات في صف واحد كبير وهذا خطأ.
- Long parameter List : يتم حله عن طريق الobjects.
- Type Tests : يتم حله عن طريق الpolymorphism وهي أحد المبادئ الأساسية في البرمجة غرضية التوجه تعني اسم واحد action مختلف عن طريق overriding وoverloading.

overloading: نفس التابع يعاد تعريفه بتمرير parameters مختلفة بالعدد، النوع، الترتيب أو الدمج بينهم بهدف زيادة المرونة بالتعامل مع الأنماط.

Overriding: تغيير body التابع ليناسب الصف الخاص بنا.

- Shotgun Surgery : تعديل بسيط يؤثر على الكثير من الobjects نتيجة خطأ في ربط الصفوف مع بعضها.

Design patterns

الأنماط التصميمية : حلول لمشاكل مكررة وهي ليست حلول كاملة بل أفكار حلول عددها 23 أساسية لكل نمط اسم هي طريقة abstract لإعادة الاستخدام وهي إعادة استخدام للمعرفة (أرقى أنواع إعادة الاستخدام).

كل نمط هو وصف للمشكلة وجوهر الحل.

- يجب أن يكون مجرد من أجل إعادة الاستخدام.
- تستخدم البرمجة غرضية التوجه بشكل أساسي pattern Elements :
  - ✓ اسم ذو مغزى.
  - ✓ وصف المشكلة.
  - ✓ وصف الحل ليس حل نهائي بل قابل للحل الذي يمكن إنشاؤه في طرق مختلفة.
  - ✓ النتائج.

بعض الخصائص :

1. تقدم حلول لمشاكل تصميمية.
2. لها أسماء Composite, Visitor, Observer .
3. لها ثلاث أنواع أساسية :
   - Behavioral : لها علاقة بالسلوك وكيف تتفاعل الأغراض.
   - Composite : لها علاقة بطريقة تجميع الأغراض.
   - Structural : لها علاقة بالبنية.
4. دائماً يوجد تفاوض وتعديلات للوصول للحل.
5. يمكن أن تنفذ بطرق مختلفة في سياقات مختلفة.

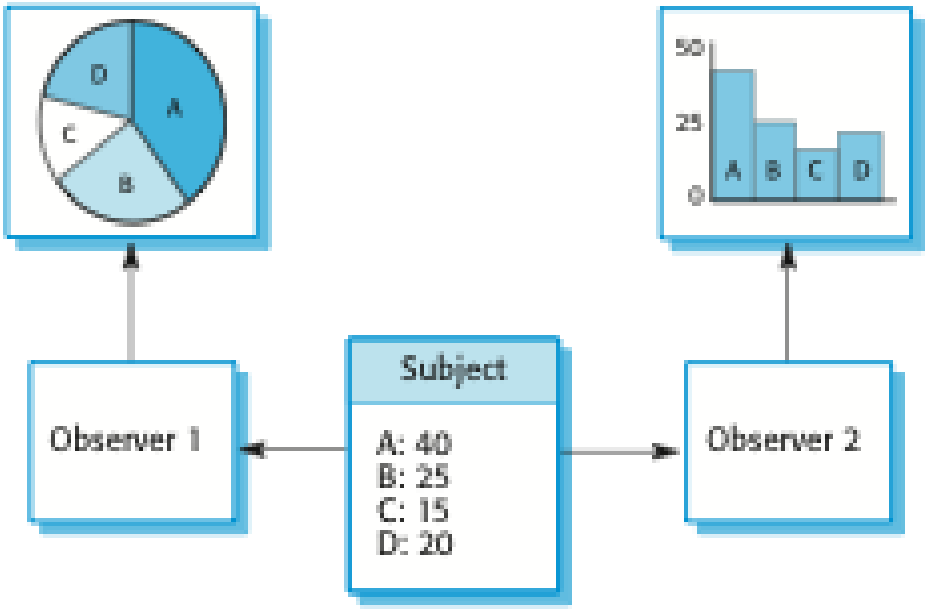
Why Design Patterns

1. **Smart** : الحل ذكي.
2. **Generic** : عام.
3. **Well-proven** : مبرهنة بشكل جيد.
4. **Simple** : بسيطة.

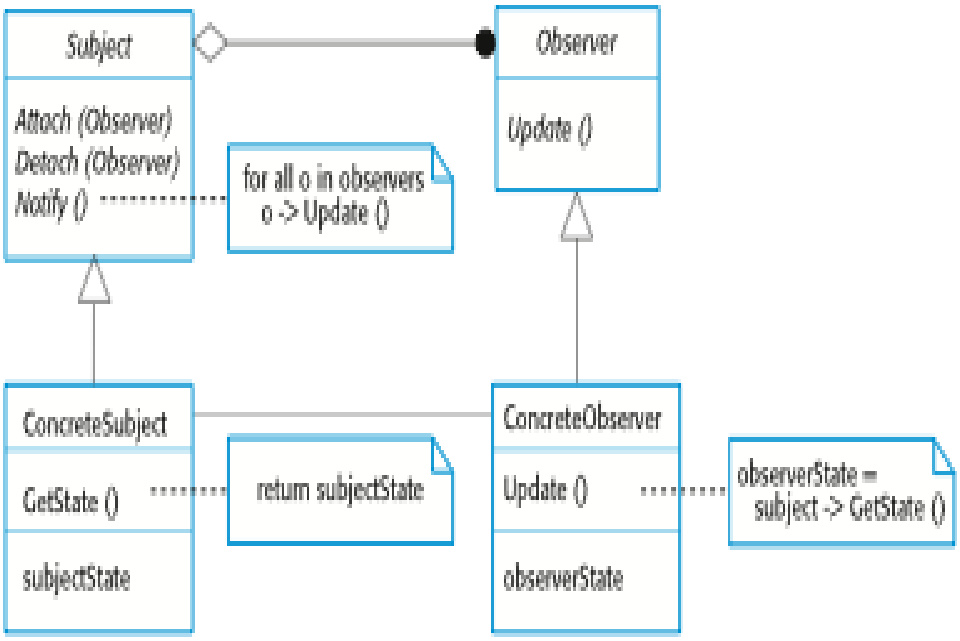
ملاحظة: ليس بالضرورة أن نستخدم **Design patterns**.

مثال **The observed Pattern**

عندما نريد عرض حالة الغرض بأكثر من طريقة (طريقة تعتمد على الدوائر، الأعمدة، ....)



تعتمد على مخططات **UML**



Implementation Issues

- Reuse خلال عملنا يجب ان ننتبه لموضوع الreusability الكود الذي نكتبة يجب أن يكون قابل للقراءة والتعديل.
- Configuration management : خلال عملية التطوير يجب تتبع الإصدارات المختلفة لكل مكون من النظام وكل تعديل يعد إصدار جديد.
- Host-target development : النظام في النهاية سيعمل على بيئة hardware فيجب أن نأخذ بعين الاعتبار القيود الفيزيائية التي تفرضها بيئة العمل

Reuse levels

- The abstraction level : أرقى أنواع إعادة الاستخدام وهو إعادة استخدام المعرفة مثل Design pattern.
- The Object level.
- The Component level.
- The System level.

كلفة إعادة الاستخدام دوماً أقل من كلفة التطوير ولكن ليس دوماً تتوفر الأجزاء التي نحتاجها.

Integrated development environment (IDEs)

هي عبارة عن بيئات تطوير جاهزة تكون مزدوجة بـtools جاهزة مثل استخدام CASE tools مما يساعد في تسريع عملية التطوير من خلال رسومات ومخططات مثل Net beans.

Open Source System

يكون الكود متوفر وقابل للتطوير أشهر مثال هو MySQL, Linux بهدف زيادة عملية التعليم وتطوير النظام.

معظم الشركات تستخدمه ويحصلون على الربح من خلال الخدمات الإضافية مثل الصيانة وإصلاح الأخطاء والتعديل.

انتهت المحاضرة ... دراسة ممتعة وحظا موفقا ^^