Designing the Modules

CH6

تصميم الوحدات

1) Design Methodology

1) منهجية التصميم

We have an abstract description of a solution to our customer's problem, a software architectural design, a plan for decomposing the design into software units and allocating the system's functional requirements to them

لدينا وصف موجز لحل مشكلة العميل ، التصميم المعماري للبرنامج ، خطة لتحليل التصميم إلى وحدات برمجية ، تخصيص المتطلبات الوظيفية للنظام لها

No distinct boundary between the end of the architecture-design phase and the start of the module-design phase

لا توجد حدود مميزة بين نهاية مرحلة تصميم البنية وبدء مرحلة تصميم الوحدة

No comparable design recipes for progressing from a software units specification to its modular design

لا يوجد طريقة للتصميم قابلة للمقارنة مع تقدم مواصفات وحدة البرامج إلى تصميم الوحدات الخاص بها

The process taken towards a final solution is not as important as the documentation produced

إن العملية التي تم اتخاذها للتوصل إلى حل نهائي ليست بنفس أهمية الوثائق التي تم إنتاجها

Design decisions are periodically revisited and revised

تتم مراجعة قرارات التصميم وتنقيحها بشكل دوري

Refactoring

إعادة بناء التعليمات البرمجية

Objective: to simplify complicated solutions or to optimize the design

الهدف: تبسيط الحلول المعقدة أو لتحسين التصميم

2) Design Principles

2) مبادئ التصميم

Design principles are guidelines for decomposing a system's required functionality and behavior into modules

مبادئ التصميم هي مبادئ توجيهية لتحليل وظائف وسلوك النظام المطلوب في الوحدات

The principles identify the criteria :

تحدد المبادئ المعايير :

- For decomposing a system
  - لتحليل النظام
- Deciding what information to provide (and what to conceal) in the resulting modules
  - تحديد المعلومات التي يجب توفيرها (والتي يجب إخفاؤها) في الوحدات الناتجة

Six dominant principles :

ستة مبادئ مهيمنة :

Modularity	Interfaces	التجزيء	واجهات
Information hiding	Incremental development	إخفاء المعلومات	التنمية المتزايدة
Abstraction	Generality	التجريد	عمومية

Modularity

التجزيء

Modularity is the principle of keeping separate the various unrelated aspects of a system, so that each aspect can be studied in isolation (also called separation of concerns)

هي مبدأ جعل فواصل مختلفة الجوانب غير مترابطة في النظام ، بحيث يمكن دراسة كل جانب بمعزل عن غيره

If the principle is applied well, each resulting module will have a single purpose and will be relatively independent of the others

إذا تم تطبيق المبدأ بشكل جيد ، فسيكون لكل وحدة ناتجة غرض واحد وستكون مستقلة نسبيًا عن الوحدات الأخرى

- each module will be easy to understand and develop

كل وحدة سيكون من السهل فهمها وتطويرها

- easier to locate faults (because there are fewer suspect modules per fault)

من السهل تحديد موقع الأخطاء (لأن هناك وحدات مشبوهة أقل)

- Easier to change the system (because a change to one module affects relatively few other modules

أسهل لتغيير النظام (لأن التغيير إلى وحدة واحدة يؤثر على عدد قليل نسبيا من الوحدات الأخرى

To determine how well a design separates concerns, we use two concepts that measure module independence: coupling and cohesion

لتحديد مدى جودة التصميم الذي يفصل بين الوحدات، نستخدم مفهومين لقياس استقلالية الوحدة : (coupling and cohesion)

Interfaces

واجهات

An **interface** defines what services the software unit provides to the rest of the system, and how other units can access those services

تحدد الواجهة ما هي الخدمات التي توفرها وحدة البرنامج لبقية النظام ، وكيف يمكن للوحدات الأخرى الوصول إلى هذه الخدمات

- For example, the interface to an object is the collection of the object's public operations and the operations' **signatures**, which specify each operation's name, parameters, and possible return values

An interface must also define what the unit requires, in terms of services or assumptions, for it to work correctly

يجب أن تحدد الواجهة أيضًا ما تتطلبه الوحدة ، من حيث الخدمات أو الافتراضات ، حتى تعمل بشكل صحيح

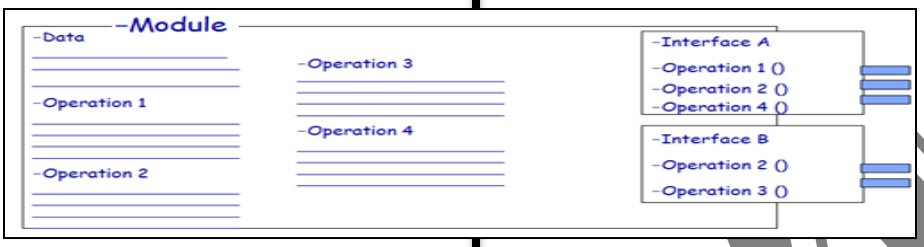
A software unit's interface describes what the unit requires of its environment, as well as what it provides to its environment

تصف واجهة وحدة البرنامج ما تطلبه الوحدة من بيئتها ، وكذلك ما توفره لبيئتها

Interfaces

A software unit may have several interfaces that make different demands on its environment or that offer different levels of service

قد يكون للوحدة البرمجية العديد من الواجهات التي تجعل المتطلبات المختلفة على بيئتها أو التي توفر مستويات مختلفة من الخدمة



The **specification** of a software unit's interface describes the externally visible properties of the software unit

توضح مواصفات واجهة وحدة البرامج الخصائص المرئية الخارجية لوحدة البرنامج

An interface specification should communicate to other system developers everything that they need to know to use our software unit correctly

يجب أن تتوصل مواصفات الواجهة بمطوري النظام الآخرين بكل ما يحتاجون إلى معرفته لاستخدام وحدة البرامج الخاصة بنا بشكل صحيح

- Purpose — الغرض منها
- Preconditions (assumptions) — الشروط المسبقة
- Protocols
- Postconditions (visible effects) — الشروط اللاحقة
- Quality attributes — علامات الجودة

3) OO Design

3) تصميم الكائنات الموجهة

Object oriented methodologies are the most popular and sophisticated design methodologies

تعد منهجية البرمجة الموجهة للكائنات منهجية التصميم الأكثر شيوعًا وتطورًا

A design is **object oriented** if it decomposes a system into a collection of runtime components called objects that encapsulate data and functionality

التصميم هو كائن موجه إذا حللت النظام إلى مجموعة من مكونات وقت التشغيل تسمى الكائنات التي تغلف البيانات والوظائف

- Objects are uniquely identifiable runtime entities that can be designated as the target of a message or request

  الكائنات عبارة عن كيانات وقتية يمكن تحديدها بشكل فريد يمكن تحديدها كهدف لرسالة أو طلب

- Objects can be composed, in that an object's data variables may themselves be objects, thereby encapsulating the implementations of the object's internal variables

  يمكن تكوين الكائنات ، حيث أن المتغيرات الكائن قد تكون هي نفسها كائنات ، وبالتالي تغليف المتغيرات الداخلية للكائن

- The implementation of an object can be reused and extended via inheritance, to define the implementation of other objects

  يمكن إعادة استخدام وتطبيق كائن عبر الوراثة ، لتحديد تنفيذ كائنات أخرى

- OO code can be polymorphic: written in generic code that works with objects of different but related types

  الكائنات الموجهة يمكن أن تكون متعددة الأشكال: مكتوبًا بشكل عام يعمل مع كائنات من أنواع مختلفة ولكنها ذات صلة

Terminology

A **class** is a software module that partially or totally implements an abstract data type

If a class is missing implementations for some of its methods, we say that it is an **abstract class**

The class definition includes **constructor** methods that spawn new object instances

Instance variables are program variables whose values are references to objects

The runtime structure of an OO system is a set of **objects**, each of which is a cohesive collection of data plus all operations for creating, reading, altering, and destroying those data

An object's data are called **attributes**, and its operations are called **methods**

An object may have multiple interfaces, each offering a different level of access to the object's data and methods

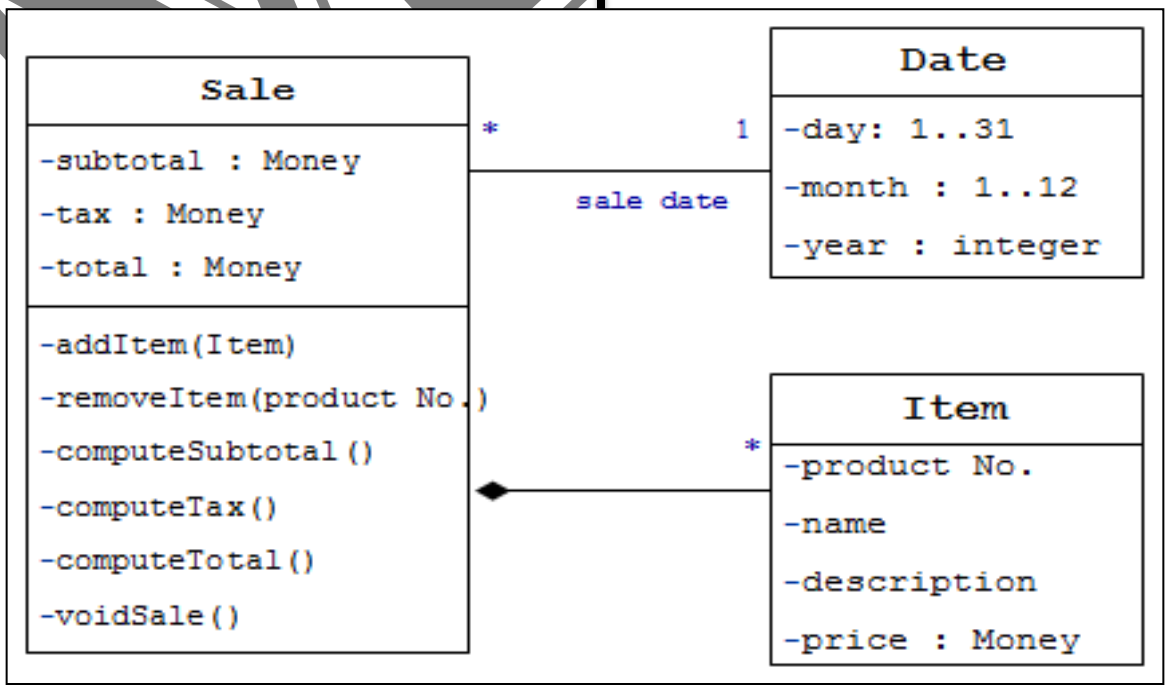
Such interfaces are hierarchically related by type: if one interface offers a strict subset of the services that another interface offers, we say that the first interface is a **subtype** of the second interface (the **super type**)

مصطلحات

(**class**) هي وحدة برامج تعمل بشكل جزئي أو كلي على تنفيذ نوع بيانات مجردة

إذا كانت إحدى الفئات تفتقد تطبيقات لبعض أساليبها ، فنحن نقول إنها (**abstract class**)

يتضمن تعريف (class) الـ (**constructor**) التي تنشئ نسخ جديدة



Terminology

Variables can refer to objects of different classes over the course of a program's execution, known as **dynamic binding**

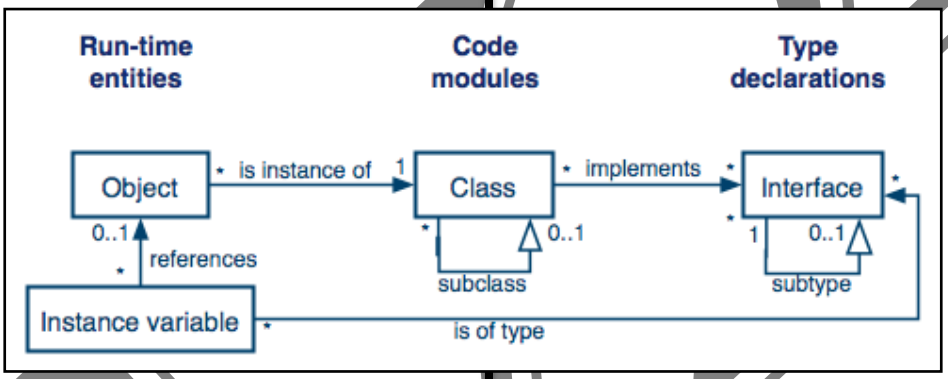
The directed arrows in the figure below depict the relationships between constructs, and the adornments at the ends of each arrow indicate the **multiplicity** (how many of an item may exist)

Four OO constructs: classes, objects, interfaces, and instance variables

مصطلحات

يمكن أن تشير المتغيرات إلى كائنات ذات فئات مختلفة على مدار تنفيذ البرنامج ، والمعروفة باسم **(dynamic binding)**

توضح الأسهم الموجهة في الشكل أدناه العلاقات بين التركيبات ، وتشير الإشارة في نهايات كل سهم إلى **(multiplicity)** (عدد العناصر التي قد توجد)



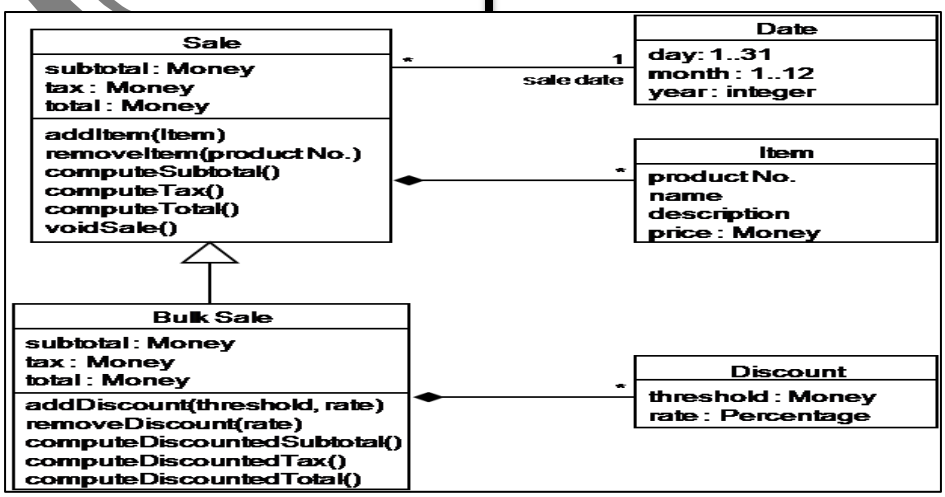
Building new classes by combining component classes, much as children build structures from building blocks is done by **object composition**

Alternatively, we can build new classes by extending or modifying definitions of existing classes

- This kind of construction, called **inheritance**, defines a new class by directly reusing (and adding to) the definitions of an existing class

Example of inheritance

بدلا من ذلك ، يمكننا بناء (classes) جديدة من خلال توسيع أو تعديل الـ (classes) الموجودة



Terminology

Polymorphism occurs when code is written in terms of interactions with an interface, but code behavior depends on the object associated with the interface at runtime and on the implementations of that object's method

Inheritance, object composition, and polymorphism are important features of an OO design that make the resulting system more useful in many ways

Inheritance vs. Object Composition

A key design decision is determining how best to structure and related complex objects

In an OO system, there are two main techniques for constructing large objects

Inheritance
Composition

A new class can be created by extending and overriding the behavior of an existing class, or it can be created by combining simpler classes to form a composite class

Each construction paradigm has advantages and disadvantages

Composition is better than inheritance at preserving the encapsulation of the reused code, because a composite object accesses the component only through its advertised interface

By contrast, using the inheritance approach, the subclass implementation is determined at design time and is static

The resulting objects are less flexible than objects instantiated from composite classes because the methods they inherit from their parent class cannot be changed at runtime

The greatest advantage of inheritance is the ability to change and specialize the behaviors of inherited methods, by selectively overriding inherited definitions ➔➔

مصطلحات

يحدث تعدد الأشكال عند كتابة التعليمات البرمجية من حيث التفاعلات مع واجهة ، ولكن سلوك الكود يعتمد على الكائن المرتبط بالواجهة في وقت التشغيل وعلى تطبيقات طريقة ذلك الكائن

الوراثة وتكوين الكائن وتعدد الأشكال هي ميزات مهمة لتصميم البرمجة الموجهة مما يجعل النظام الناتج أكثر فائدة بطرق عديدة

الوراثة مقابل تكوين الكائن

قرار التصميم الرئيسي هو تحديد أفضل طريقة للبناء والأشياء المعقدة ذات الصلة

في نظام البرمجة الموجهة ، هناك طريقتان رئيسيتان لبناء الأجسام الكبيرة

يمكن إنشاء (class) جديدة من خلال توسيع وتجاوز سلوك (class) موجودة ، أو يمكن إنشاؤها من خلال الجمع بين (classes) الأبسط لتشكيل فئة مركبة

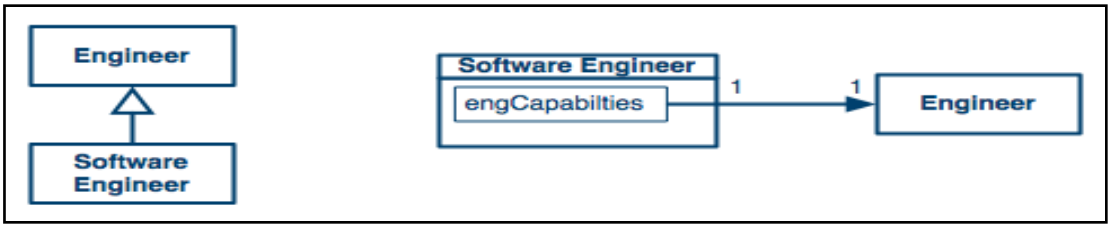
كل نموذج بناء له مزايا وعيوب

التركيب أفضل من الوراثة في الحفاظ على تغليف الشفرة المعاد استخدامها ، لأن الكائن المركب لا يصل إلى المكون إلا من خلال واجهته المعلنة

على النقيض من ذلك ، باستخدام نهج الوراثة ، يتم تحديد تطبيق الفئة الفرعية في وقت التصميم وهو ثابت

تكون الكائنات الناتجة أقل مرونة من الكائنات التي تم إنشاء مثيل لها من فئات مركبة لأن الطرق التي يرثونها من الصف الأصلي لا يمكن تغييرها في وقت التشغيل

أكبر ميزة في الوراثة هي القدرة على تغيير وتخصص سلوكيات الأساليب الموروثة ، من خلال تجاوز التعريفات الموروثة بشكل انتقائي



4) Design Documentation

The details of the system architecture is documented in *Software Architecture Document* (SAD)

SAD serves as a bridge between the requirements and the design

Many ways to document the design

Design by contract: a particular approach that uses the documentation only to capture the design but also to encourage interaction among developers

Design by Contract

In design by contract, each module has an interface specification that precisely describes what the module is supposed to do

- Meyer (1997) suggests that design by contract helps ensure that modules interoperate correctly
- This specification, called a **contract**, governs how the module is to interact with other modules and systems
- Such specification cannot guarantee a module's correctness, but it forms a clear and consistent basis for testing and verification
- The contract covers mutual obligations (the reconditions), benefits (the postconditions), and consistency constraints (called **invariants**)
- Together, these contract properties are called **assertions**

Design contract between software provider and user

4) وثائق التصميم

تفاصيل بنية النظام موثقة في وثيقة هندسة البرمجيات

SAD : بمثابة جسر بين المتطلبات والتصميم

العديد من الطرق لتوثيق التصميم

التصميم بالتعاقد : نهج خاص يستخدم الوثائق فقط لالتقاط التصميم ولكن أيضًا لتشجيع التفاعل بين المطورين

التصميم بالتعاقد

في التصميم حسب العقد ، تحتوي كل وحدة على مواصفات الواجهة التي تصف بدقة ما من المفترض أن تقوم به الوحدة

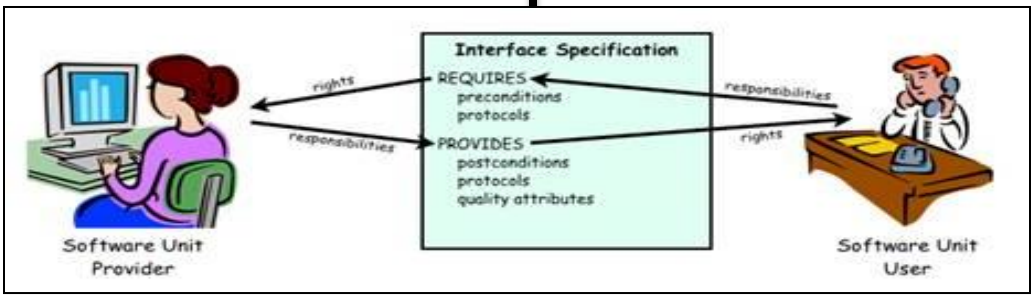
وتحدد هذه المواصفة ، و التي تسمى العقد ، كيفية تفاعل الوحدة مع الوحدات والنظم الأخرى

ولا تضمن مثل هذه المواصفات سلامة الوحدة ، ولكنها تشكل أساسًا واضحًا وثابتًا للاختبار والتحقق

يغطي العقد الالتزامات المتبادلة (شروط الاستبدال) ، والفوائد (الشروط المسبقة) ، وقيود الاتساق (تسمى الثوابت)

معا ، تسمى خصائص العقد هذه التأكيدات

عقد تصميم بين مزود البرنامج والمستخدم



5) What This Chapter Means For You

The design process describes the system components using a common language with which to communicate

Object orientation is a particularly appealing basis for such design, because it allows us to describe and reason about the system from its birth to its delivery in the same terms: classes, objects, and methods

Consistent notation makes it easier for your team to understand the implications of using a particular object or class

- Consistency assists the maintainers and testers, enabling them to build test cases and monitor changes more easily
- Because the requirements, design, and code are expressed in the same way, it is easier for you to evaluate the effects of proposed changes to the requirements or designs

5) ماذا يعني هذا الفصل بالنسبة لك

تصف عملية التصميم مكونات النظام باستخدام لغة مشتركة يتم الاتصال بها

التوجه الكائن هو أساس جذاب بشكل خاص لمثل هذا التصميم ، لأنه يسمح لنا بوصف النظام والسبب فيه من بدايته وحتى تسليمه بنفس المصطلحات :-
(classes, objects, and methods)

يسهل التدوين المتناسق على فريقك فهم الآثار المترتبة على استخدام
(particular object or class)

- يساعد التناسق بين المصممين والمشرفين ، مما يمكنهم من بناء حالات الاختبار ومراقبة التغييرات بسهولة أكبر
- نظرًا لأنه يتم التعبير عن المتطلبات والتصميم والتعليمات البرمجية بنفس الطريقة ، يكون من الأسهل عليك تقييم تأثيرات التغييرات المقترحة على المتطلبات أو التصميمات

Chapter 6 END

انتهى الفصل السادس