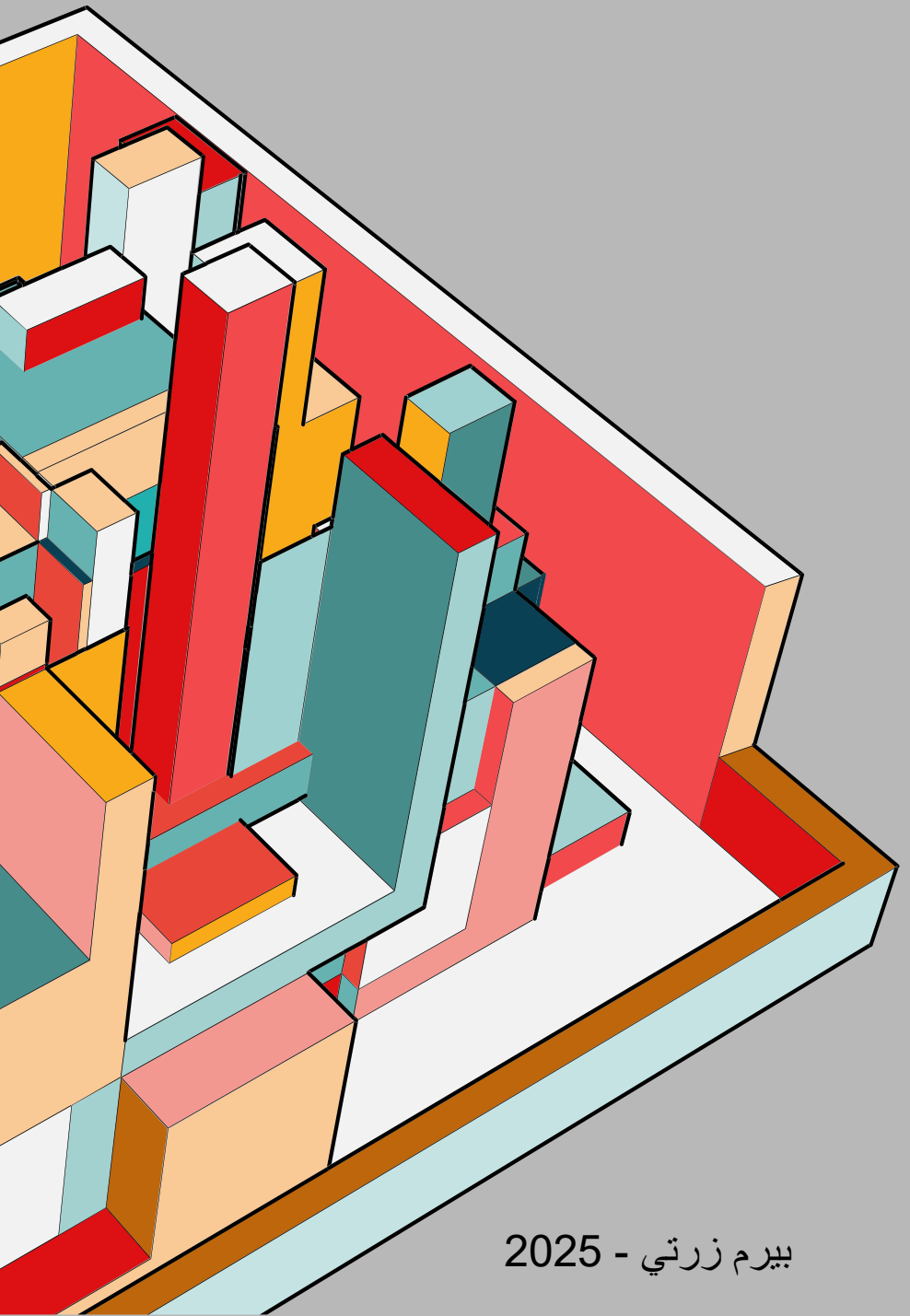


تصميم وهيكلة البرمجيات
ITSE411

المحاضرة الخامسة

Object-oriented design using UML

بيرم زرتي - 2025

SOFTWARE DESIGN

- **Detailed design:** the process of refining and expanding the preliminary design of a system or component to the extent that the design is sufficiently complete to be implemented. [ISO/IEC 24765]
- Whereas the software architecture places a major emphasis on quality (nonfunctional requirements), the detailed design activity places a major focus on addressing functional requirements of the system.
- In object-oriented systems, the detailed design activity is where components are refined into one or more classes, interfaces are realized, relationships between classes are specified, class functions and variable names are created, design patterns are identified and applied.

عرّف التصميم التفصيلي بأنه عملية **تنقيح وتوسيع التصميم الأولي** لنظام أو مكوّن برمجي، إلى الحد الذي يُصبح فيه التصميم مكتملًا بما يكفي ليتم تنفيذه فعليًا، وذلك وفقًا لتعريف معيار ISO/IEC 24765

- **التصميم المعماري** Software Architecture يُركّز بشكل أساسي على **متطلبات الجودة** أو ما يُعرف بالمتطلبات غير الوظيفية (مثل الأداء، الأمان، القابلية للتوسّع).
- **التصميم التفصيلي** Detailed Design يُركّز على **المتطلبات الوظيفية** للنظام، أي ما يجب أن يفعله النظام من وظائف محددة.

في سياق البرمجة الشيئية ، يتضمن التصميم التفصيلي ما يلي:

- **تحويل المكوّنات** إلى فئات Classes محددة.
- **تحقيق الواجهات** Interfaces وتحديد كيفية تفاعل الكائنات.
- **تحديد العلاقات** بين الفئات (مثل الوراثة، التجميع، الربط).
- **تصميم الوظائف والمتغيرات** داخل كل فئة.
- **تحديد وتطبيق أنماط التصميم** Design Patterns المناسبة لتحسين البنية والمرونة.

SOFTWARE DESIGN

What is Detailed design ?

- Two major tasks of the detailed design activity are interface design and component design.
- Interface design refers to the design activity that deals with specification of interfaces between components in the design (Sommerville 2010).
- Component design refers to modeling the internal structure and behavior of components which includes the internal structure of both logical and physical components-identified during the software architecture phase.

يُركّز التصميم التفصيلي في هندسة البرمجيات على تحويل التصور المعماري إلى تصميم قابل للتنفيذ، ويشمل مهمتين رئيسيتين:

1 تصميم الواجهات Interface Design

يشير إلى النشاط الذي يُعنى **بتحديد واجهات التفاعل بين المكوّنات** داخل النظام. ويشمل ذلك:

- تحديد طرق الاتصال بين الوحدات البرمجية.
- تعريف البروتوكولات والمعايير المستخدمة لتبادل البيانات.
- ضمان التوافق بين المكوّنات المختلفة.

2. تصميم المكوّنات Component Design

يتعلّق **بنمذجة البنية الداخلية وسلوك المكوّنات**، ويشمل:

- تحديد الهيكل الداخلي للمكوّنات المنطقية والفيزيائية التي تم تعريفها خلال مرحلة التصميم المعماري.
- تصميم الوظائف الداخلية، المتغيرات، وآليات التنفيذ.
- توضيح كيفية تفاعل كل مكوّن مع الأحداث أو البيانات الواردة.

SOFTWARE DESIGN

What is Detailed design ?

- Components are not limited to object-oriented systems; therefore, component designs can be realized in many ways. In object-oriented systems, the internal structure of components is typically modeled using UML through one or more diagrams, including class and sequence diagrams (Carlos E. Otero 2012).
- When modeling the internal structure of components, several design principles, heuristics, and patterns are used to create and evaluate component designs.

لا تقتصر المكوّنات البرمجية على الأنظمة الكائنية أو الشيئية Object-Oriented Systems، لذا يمكن **تحقيق تصميم المكوّنات** بطرق متعددة وفقًا للنموذج البرمجي المستخدم. في الأنظمة الكائنية، يتم عادةً **نمذجة البنية الداخلية للمكوّنات** باستخدام لغة النمذجة الموحدة UML ، وذلك من خلال مخططات متعددة، أبرزها:

- **مخططات الفئات** Class Diagrams تُظهر البنية الثابتة للمكوّنات، بما في ذلك الفئات، الخصائص، والوظائف.
- **مخططات التسلسل** Sequence Diagrams تُستخدم لتمثيل التفاعلات الديناميكية بين الكائنات عبر الزمن.

عند تصميم البنية الداخلية للمكوّنات، يتم الاستعانة بعدد من:

- **مبادئ التصميم** مثل مبدأ المسؤولية الواحدة
- **الاستدلالات** قواعد إرشادية تُستخدم لتوجيه القرارات التصميمية.
- **أنماط التصميم** Design Patterns حلول مجرّبة لمشكلات تصميم شائعة

UNIFIED MODELING LANGUAGE

- Unified Modeling Language (UML) is a standardized visual language used in software engineering to represent, design, and communicate the structure and behavior of software systems. It provides a common framework for software developers, architects, and stakeholders to create and understand visual models of software systems
- UML supports both structural modeling, which focuses on the static structure of the system (e.g., classes, objects, relationships), and behavioral modeling, which captures the dynamic aspects of the system

تُعد **لغة النمذجة الموحدة** UML لغة بصرية معيارية تُستخدم في هندسة البرمجيات لتمثيل وتصميم والتواصل حول بنية وسلوك الأنظمة البرمجية. وتُوفّر إطارًا مشتركًا يمكّن المطوّرين والمعماريين وأصحاب المصلحة من **إنشاء وفهم النماذج البصرية** للنظام البرمجي.

أنواع النمذجة في UML

النمذجة الهيكلية Structural Modeling

تركّز على **البنية الثابتة للنظام**، وتشمل:

- الفئات Classes
- الكائنات Objects
- العلاقات Relationships مثل الوراثة والتجميع

النمذجة السلوكية Behavioral Modeling

تهدف إلى **تمثيل الجوانب الديناميكية للنظام**، مثل:

- التفاعلات بين الكائنات
- تسلسل الأحداث
- حالات النظام واستجابته للمدخلات

UNIFIED MODELING LANGUAGE (UML)

- Unified Modeling Language (UML) diagrams are :

STRUCTURAL

- CLASS DIAGRAM
- OBJECT DIAGRAM
- COMPONENT DIAGRAM
- DEPLOYMENT DIAGRAM

BEHAVIORAL

- USE CASE DIAGRAM
- SEQUENCE DIAGRAM
- COLLABORATION DIAGRAM
- STATE CHART DIAGRAM
- ACTIVITY DIAGRAM

STRUCTURAL

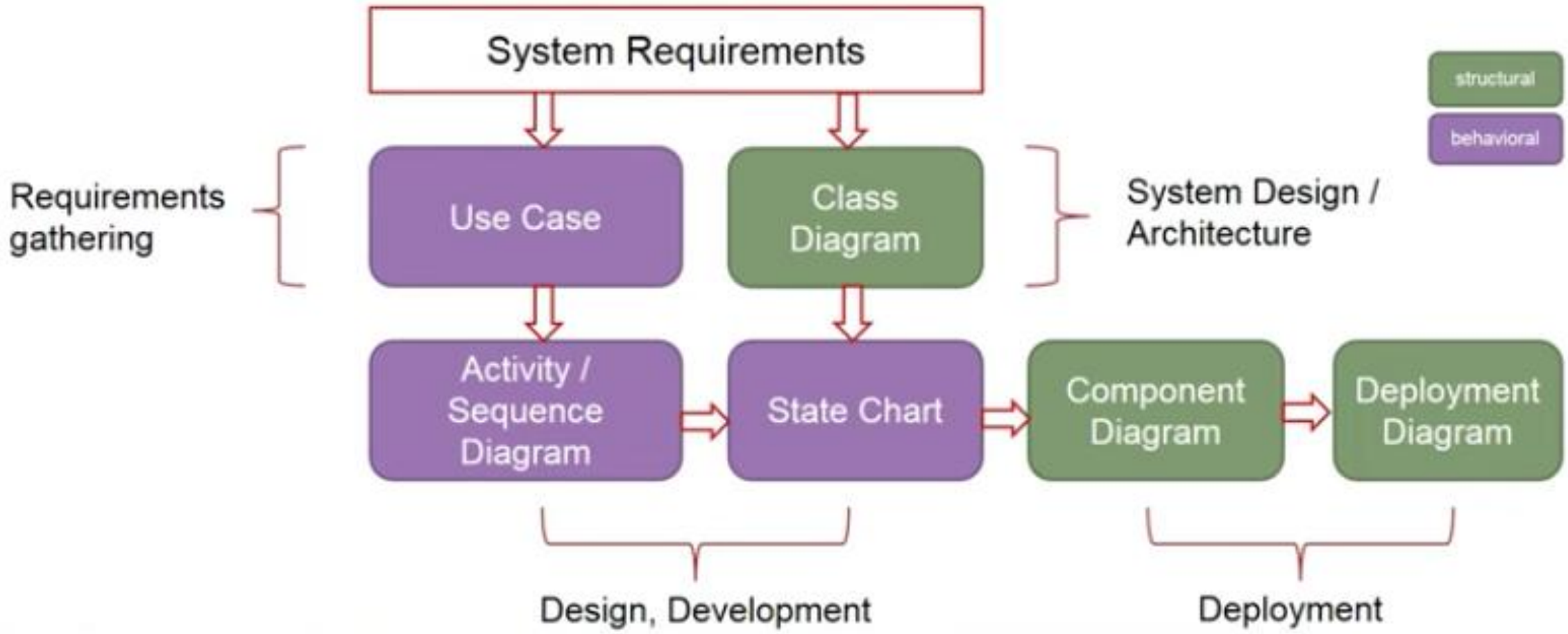
المخطط	الوصف
Class Diagram	يُظهر الفئات Classes وخصائصها وعلاقاتها مثل الوراثة والتجميع.
Object Diagram	يُمثّل الكائنات Objects في لحظة معينة، ويُظهر حالتها وعلاقاتها.
Component Diagram	يُوضّح مكوّنات النظام البرمجية وكيفية تجميعها وتفاعلها.
Deployment Diagram	يُبيّن كيفية نشر المكوّنات على الأجهزة أو العقد الفيزيائية.

BEHAVIORAL

المخطط	الوصف
Use Case Diagram	يُظهر التفاعلات بين المستخدمين Actors والنظام من خلال حالات الاستخدام.
Sequence Diagram	يُوضّح تسلسل الرسائل بين الكائنات لتحقيق وظيفة معينة.
Collaboration Diagram	يُركّز على العلاقات بين الكائنات وتعاونها لتنفيذ سلوك معين.
State Chart Diagram	يُمثّل حالات الكائنات وكيفية انتقالها من حالة لأخرى بناءً على الأحداث.
Activity Diagram	يُظهر تدفق الأنشطة والعمليات داخل النظام، ويُستخدم لنمذجة منطق الأعمال.

UNIFIED MODELING LANGUAGE (UML)

- Unified Modeling Language (UML) diagrams are :



GENERAL DIAGRAMMING GUIDELINES

The guidelines presented are applicable to all types of diagrams, UML or otherwise.

❖ Readability Guidelines

1. Avoid Crossing Lines

When two lines cross on a diagram, such as two associations on a UML class diagram, the potential for misreading a diagram exists.

2. Depict Crossing Lines as a Jump

You can’ t always avoid crossing lines; for example, you cannot fully connect five symbols. When you need to have two lines cross, one of them should “hop” over the other as in Figure 1:

نطبق الإرشادات التالية على جميع أنواع المخططات، بما في ذلك مخططات UML، وتهدف إلى تعزيز وضوح التصميم وتقليل احتمالية سوء الفهم:

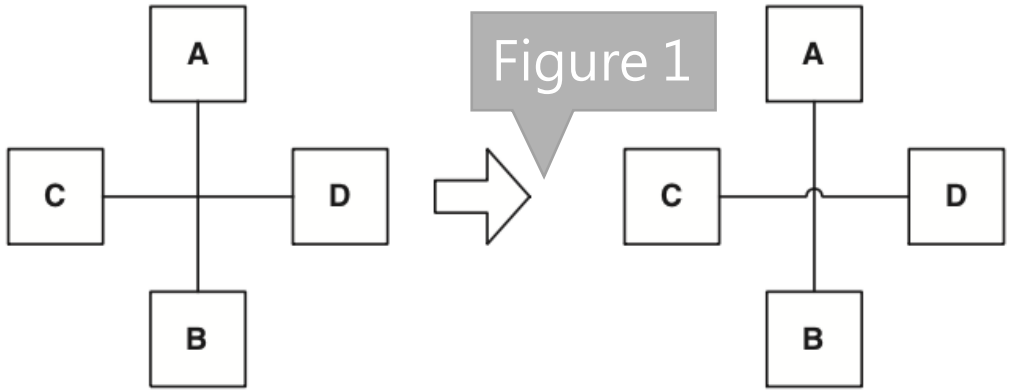
1. 🚫 تجنّب تقاطع الخطوط

عند تقاطع خطّين في المخطط، مثل تقاطع علاقات في مخطط الفئات Class Diagram، تزداد احتمالية تفسير المخطط بشكل خاطئ.

لذلك يُفضّل ترتيب العناصر بطريقة تُقلّل من هذه التقاطعات.

2. ⤴️ تمثيل الخطوط المتقاطعة

في حال كان من الضروري أن تتقاطع الخطوط، يُوصى باستخدام تقنية "القفز" Jump لتمييز أحد الخطين عن الآخر بصريًا، مما يُساعد في توضيح أن الخطين لا يرتبطان مباشرة ويُقلّل من الالتباس.



GENERAL DIAGRAMMING GUIDELINES

3. Avoid Diagonal or Curved Lines:

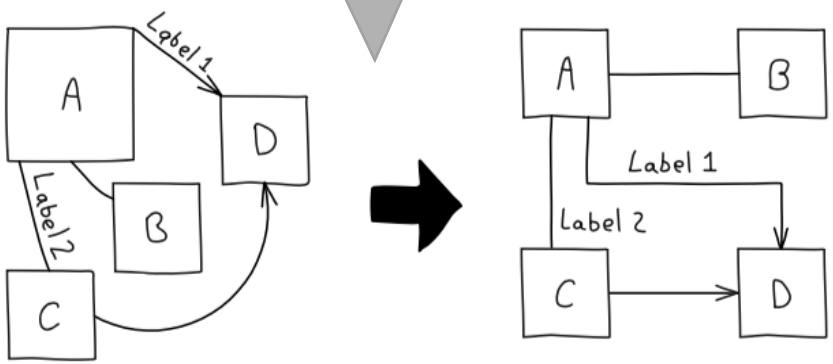
Straight lines, drawn either vertically or horizontally, are easier for your eyes to follow than diagonal or curved lines

4. Apply Consistently Sized Symbols

5. Align Labels Horizontally

In Figure 2 the two labels are easier to read in the second version of the diagram. Notice how Label 2 is horizontal even though the line it is associated with is vertical.

Figure 2



3. ➖ **تجنّب الخطوط المائلة أو المنحنية** يُفضّل استخدام **الخطوط المستقيمة** سواء كانت رأسية أو أفقية، لأنها أسهل للعين في التتبّع مقارنة بالخطوط المائلة أو المنحنية، مما يُحسّن من وضوح المخطط وسهولة فهمه.

4. ⬜ **استخدام رموز بحجم موحّد** يُوصى باستخدام رموز متساوية الحجم في جميع أنحاء المخطط، مما يُعزّز التناسق البصري ويُسهّل التمييز بين العناصر المختلفة دون تشويش بصري.

5. 🏷️ **محاذاة التسميات أفقيًا**

يُفضّل أن تكون **التسميات مكتوبة بشكل أفقي** حتى لو كانت مرتبطة بخط رأسي، لأن النص الأفقي أسهل في القراءة. كما أن المحاذاة الأفقية تُضفي مظهرًا أكثر تنظيمًا على المخطط.

في الشكل التوضيحي Figure 2، يظهر الفرق بوضوح بين النسختين؛ حيث تكون التسميات في النسخة الثانية أكثر قابلية للقراءة بفضل المحاذاة الأفقية.

GENERAL DIAGRAMMING GUIDELINES

6. Organize Diagrams Left to Right, Top to Bottom

7. Avoid Many Close Lines:: Several lines close together are hard to follow.

8. Reorganize Large Diagrams into Several Smaller Ones

A good rule of thumb is that a diagram shouldn’ t have more than nine symbols on it, based on the 7 ± 2 rule (Miller 1957),

6. ↔️ تنظيم المخططات من اليسار إلى اليمين ومن الأعلى إلى الأسفل

يُفضّل ترتيب عناصر المخطط بطريقة تُحاكي أسلوب القراءة الطبيعي، مما يُسهّل تتبّع العلاقات والتسلسل المنطقي للمحتوى.

7. 🚫 تجنّب كثافة الخطوط المتقاربة

وجود عدة خطوط متقاربة يُصعّب تتبّعها بصريًا، وقد يؤدي إلى التباس في فهم العلاقات. يُوصى بتوفير مسافات كافية بين الخطوط.

8. 🧩 إعادة تنظيم المخططات الكبيرة إلى عدة مخططات صغيرة

قاعدة عامة: لا ينبغي أن يحتوي المخطط على أكثر من **تسعة رموز**، استنادًا إلى قاعدة "7 ± 2" التي اقترحها Miller (1957)، والتي تُشير إلى حدود الإدراك البشري. تقسيم المخطط يُسهّل الفهم ويُقلّل من التعقيد.

GENERAL DIAGRAMMING GUIDELINES

9. Prefer Single-Page Diagrams

a diagram should be printable on a single sheet of paper to help reduce its scope as well as to prevent wasted time cutting and taping several pages together.

10. Focus on Content First, Appearance Second

11. Apply Consistent, Readable Fonts

Consistent, easy-to-read fonts improve the readability of your diagrams. Good ideas include fonts in the Courier, Arial, and Times families. Bad ideas include small fonts (less than 10 point), large fonts (greater than 18 point), and italics.

9. 📄 تفضيل المخططات ذات الصفحة الواحدة

يُفضّل أن يكون المخطط قابلًا للطباعة على ورقة واحدة، مما يُقلّل من نطاقه ويُجنّب الحاجة إلى قصّ وتجميع عدة صفحات.

10. 🎯 التركيز على المحتوى أولًا، ثم المظهر

يجب أن يكون الهدف الأساسي من المخطط هو **نقل المعلومات بوضوح**، ثم يُمكن تحسين المظهر البصري لاحقًا دون التأثير على دقة المحتوى.

11. 🔤 استخدام خطوط موحّدة وسهلة القراءة

اختيار الخطوط المناسبة يُعزّز من وضوح المخطط، ويُوصى باستخدام خطوط مثل:

- Courier
- Arial
- Times

يُفضّل تجنّب:

- الخطوط الصغيرة (أقل من 10 نقطة)
- الخطوط الكبيرة جدًا (أكثر من 18 نقطة)
- الخطوط المائلة Italics

GENERAL DIAGRAMMING GUIDELINES

12. Apply Common Domain Terminology in Names

Apply consistent and recognizable domain terminology, such as customer and order, whenever possible on your diagrams.

13. Name Common Elements Consistently Across Diagrams

A single modeling element, such as an actor or a class, will appear on several of your diagrams. For example, the same class will appear on several UML class diagrams, several UML sequence diagrams, several UML communication diagrams, and several UML activity diagrams. This class should have the same name on each diagram; otherwise your readers will become confused.

12. استخدام مصطلحات المجال الشائعة

يُوصى باستخدام **مصطلحات مألوفة ومتسقة من مجال النظام** عند تسمية العناصر داخل المخططات، مثل "عميل" Customer، "طلب" Order ، أو "فاتورة" Invoice هذا يُسهّل على المطوّرين وأصحاب المصلحة فهم المخطط وربطه بسياق الأعمال الحقيقي.

13. تسمية العناصر المشتركة بشكل موحّد عبر المخططات

عند تكرار عنصر نمذجة مثل "فئة" Class أو "ممثل" Actor في عدة مخططات UML مثل مخطط الفئات، التسلسل، التعاون، النشاط ، يجب الحفاظ على **نفس الاسم** لهذا العنصر في جميع المخططات. عدم الاتساق في التسمية قد يُسبب ارتباكًا ويُضعف من وضوح النموذج العام.

GENERAL DIAGRAMMING GUIDELINES

A UML note is a modeling construct for adding textual information—such as a comment, constraint definition, or method body—to UML diagrams. As you can see in Figure below, notes are depicted as rectangles with the top right corners folded over.

14. Left-Justify Text in Notes

It is common practice to left-justify text in UML notes, as you can see in Figure 3.

Figure 3

الملاحظات في UML (UML Notes)

تُعد **الملاحظة** Note في لغة النمذجة الموحدة UML عنصرًا نمذجيًا يُستخدم لإضافة معلومات نصية إلى المخططات. وتشمل هذه المعلومات:

- التعليقات التوضيحية
- تعريف القيود Constraints
- محتوى الوظائف أو الإجراءات Method Body

تُعرض الملاحظات في UMLعلى شكل **مستطيل** مع **انحناء في الزاوية العلوية اليمنى**، مما يُميّزها عن باقي العناصر في المخطط.

تُستخدم الملاحظات لتوضيح أو توثيق أجزاء من النموذج دون التأثير على منطق النظام، وهي مفيدة في التواصل بين المطوّرين وأصحاب المصلحة.

14. محاذاة النص إلى اليسار داخل الملاحظات من **الممارسات الشائعة** في تصميم مخططات UMLأن يتم **محاذاة النص داخل الملاحظات إلى اليسار**. هذا الأسلوب يُسهّل قراءة المحتوى النصي، سواء كان تعليقًا، تعريفًا لقيد، أو وصفًا لطريقة تنفيذ.

حتى عندما تحتوي الملاحظة على عدة أسطر أو معلومات تقنية، فإن المحاذاة اليسارية تُحافظ على التناسق البصري وتُعزّز قابلية القراءة.

GENERAL DIAGRAMMING GUIDELINES

15. Prefer Notes over OCL to Indicate Constraints:

In UML, constraints are modeled either by a UML note using free-form text or with Object Constraint Language (OCL).

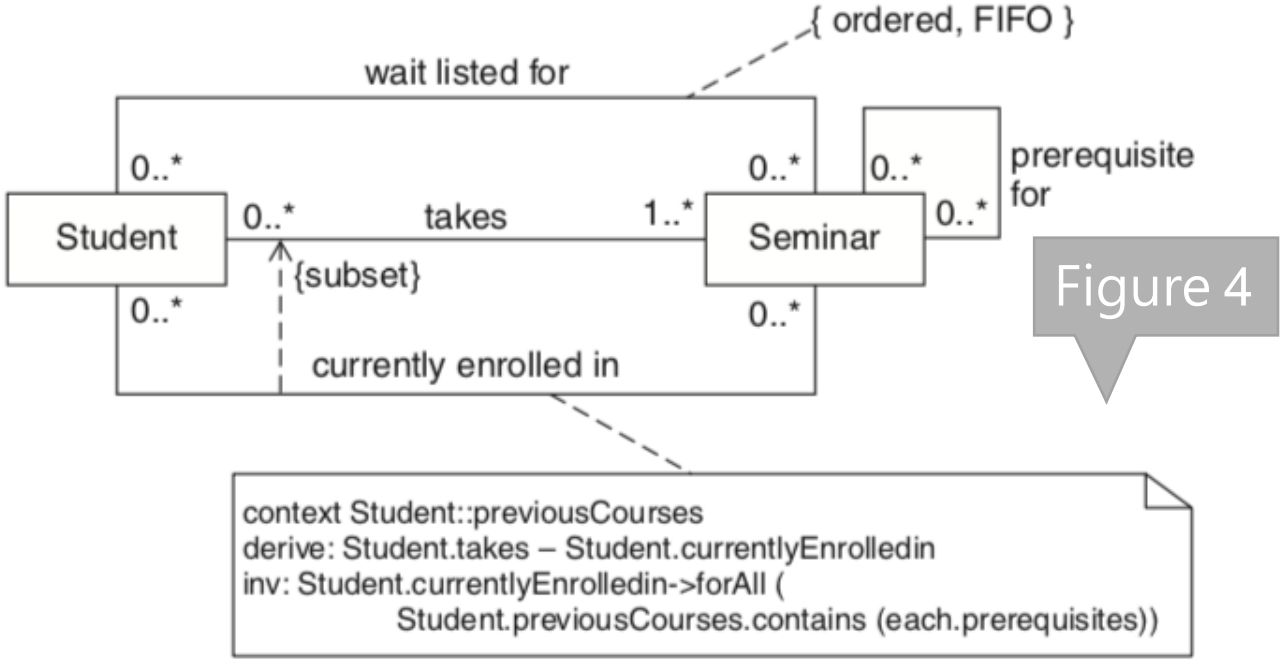


Figure 4

15. تفضيل استخدام الملاحظات على لغة القيود الكائنية OCL لتمثيل القيود

في UML، يمكن **تمثيل القيود** Constraints بطريقتين:

1.باستخدام **ملاحظة** (Note) UMLتحتوي على نص حر يوضّح القيد المطلوب.

2.باستخدام **لغة القيود الكائنية** - Object Constraint Language OCL، وهي لغة رسمية تُستخدم لتحديد القيود بدقة رياضية.

يُفضّل استخدام **الملاحظات النصية** بدلًا من OCL في معظم الحالات، لأنها:

- أكثر وضوحًا وسهولة في الفهم لغير المختصين.
- تُسهّل التواصل بين المطوّرين وأصحاب المصلحة.
- تُقلّل من تعقيد المخطط وتُحافظ على قابليته للقراءة.

تُستخدم OCLفي الحالات التي تتطلب دقة عالية أو تحقق آلي للقيود، بينما تُعد الملاحظات خيارًا عمليًا ومرنًا في التصميم اليومي.

UML USE-CASE DIAGRAMS

A UML use-case diagram shows the relationships among actors and use cases within a system. They are often used to:

- ✓ provide an overview of all or part of the usage requirements for a system or organization in the form of an essential model.
- ✓ model the analysis of usage requirements in the form of a system use-case model.
- ✓ The use case model describes the functional requirements of the system in terms of the actors and use cases

يُستخدم **مخطط حالات الاستخدام** في لغة النمذجة الموحدة UML لتمثيل **العلاقات بين الممثلين** Actors وحالات الاستخدام Use Cases داخل النظام.

الأهداف الرئيسية لمخطط حالات الاستخدام:

- **تقديم نظرة شاملة** على جميع أو بعض متطلبات الاستخدام لنظام أو مؤسسة، وذلك في شكل نموذج جوهري Essential Model .
- **نمذجة تحليل متطلبات الاستخدام** من خلال بناء نموذج حالات الاستخدام للنظام System Use-Case Model

يُعبّر نموذج حالات الاستخدام عن **المتطلبات الوظيفية للنظام** من خلال:

- تحديد الممثلينActors مثل المستخدمين أو الأنظمة الخارجية التي تتفاعل مع النظام.
- تحديد حالات الاستخدام Use Cases وهي الوظائف أو السيناريوهات التي يُنفّذها النظام استجابة لتفاعل الممثلين.

يُعد هذا النوع من المخططات أداة فعّالة في مرحلة تحليل النظام، حيث يُساعد على فهم ما يجب أن يفعله النظام من وجهة نظر المستخدم النهائي.

IDENTIFYING USE CASES

A use case defines a sequence of interactions between one or more actors and the system.

A use case always starts with input from an actor. A use case typically consists of a sequence of interactions between the actor and the system. Each interaction consists of an input from the actor followed by a response from the system.

In this way, the functional requirements of the system are described in terms of the use cases

تُعرّف **حالة الاستخدام** Use Case بأنها تسلسل من التفاعلات بين **ممثل واحد أو أكثر** Actors والنظام، بهدف تحقيق وظيفة محددة ضمن النظام.

آلية التفاعل:

- تبدأ حالة الاستخدام دائمًا **بمدخل من الممثل** (مثل مستخدم أو نظام خارجي).
- تتكوّن الحالة عادةً من **سلسلة من التفاعلات**، حيث يُقدّم الممثل مدخلًا، ويستجيب النظام بنتيجة أو إجراء.
- يتكرّر هذا النمط من التفاعل حتى يتم تنفيذ الوظيفة المطلوبة بالكامل.

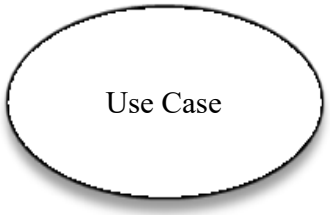
العلاقة بالمتطلبات الوظيفية:

من خلال حالات الاستخدام، يتم **وصف المتطلبات الوظيفية للنظام** بطريقة عملية، حيث تُوضّح كيف يتفاعل المستخدمون مع النظام لتحقيق أهدافهم.

تُعد حالات الاستخدام أداة تحليلية قوية لفهم سلوك النظام من وجهة نظر المستخدم، وتُستخدم على نطاق واسع في مراحل تحليل وتصميم البرمجيات.

COMPONENTS OF USE CASE DIAGRAMS

Use Case
Functionality Or Services Provided By The System



حالات Use Cases
الاستخدام: الوظائف أو السيناريوهات التي يقدّمها النظام للمستخدم. مثال: "تسجيل الدخول"، "شراء منتج"، "استعارة كتاب."

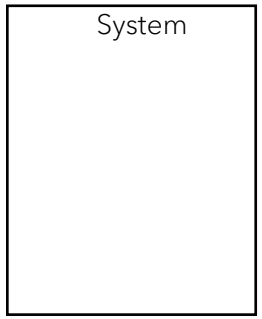
Actors
Who Interacts With The System

Actors
الممثلون/المستخدمون : الأشخاص أو الأنظمة الخارجية التي تتفاعل مع النظام. مثال: "الزبون"، "أمين المكتبة"، أو "نظام دفع إلكتروني

Relationship
Relation Between Actors and use cases

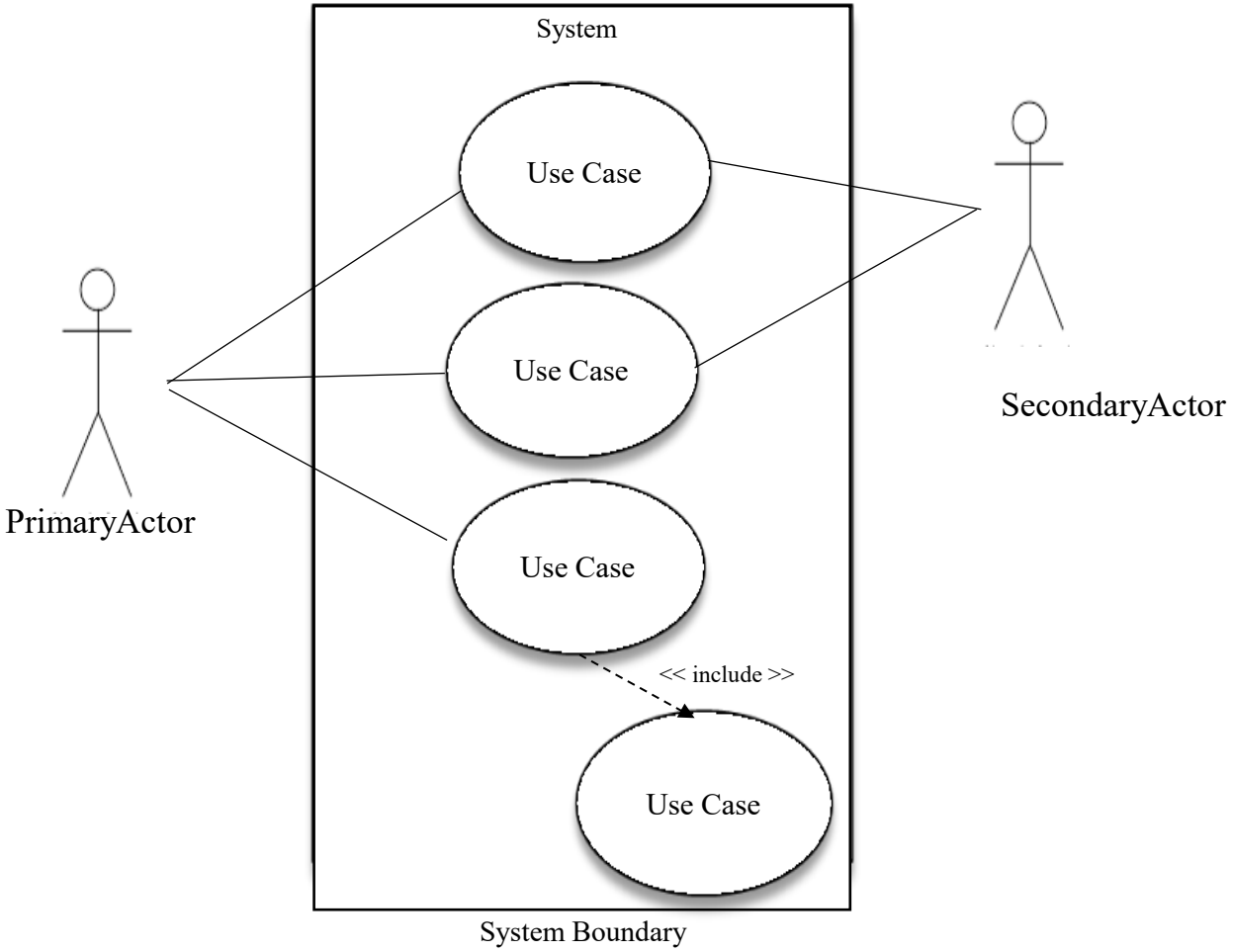
Relationships
العلاقات الروابط بين الممثلين وحالات الاستخدام، أو بين حالات الاستخدام نفسها.

System Boundary
it indicates the scope of the system



System Boundary ."حدود
النظام إطار يوضح ما هو داخل النظام وما هو خارجه. يرسم عادةً كمستطيل يحتوي على حالات الاستخدام.

COMPONENTS OF USE CASE DIAGRAMS



مثلا 1 نظام مكتبة:

Actor:

طالب، أمين مكتبة.

Use Cases:

استعارة كتاب، إرجاع كتاب، إضافة كتاب جديد.

System Boundary

نظام المكتبة.

Relationships:

الطالب ↔ استعارة كتاب، أمين المكتبة ↔ إضافة كتاب جديد.

IDENTIFYING USE CASES

Let us consider the banking example, The customer can initiate three use cases: Withdraw Funds, Query Account, and Transfer Funds

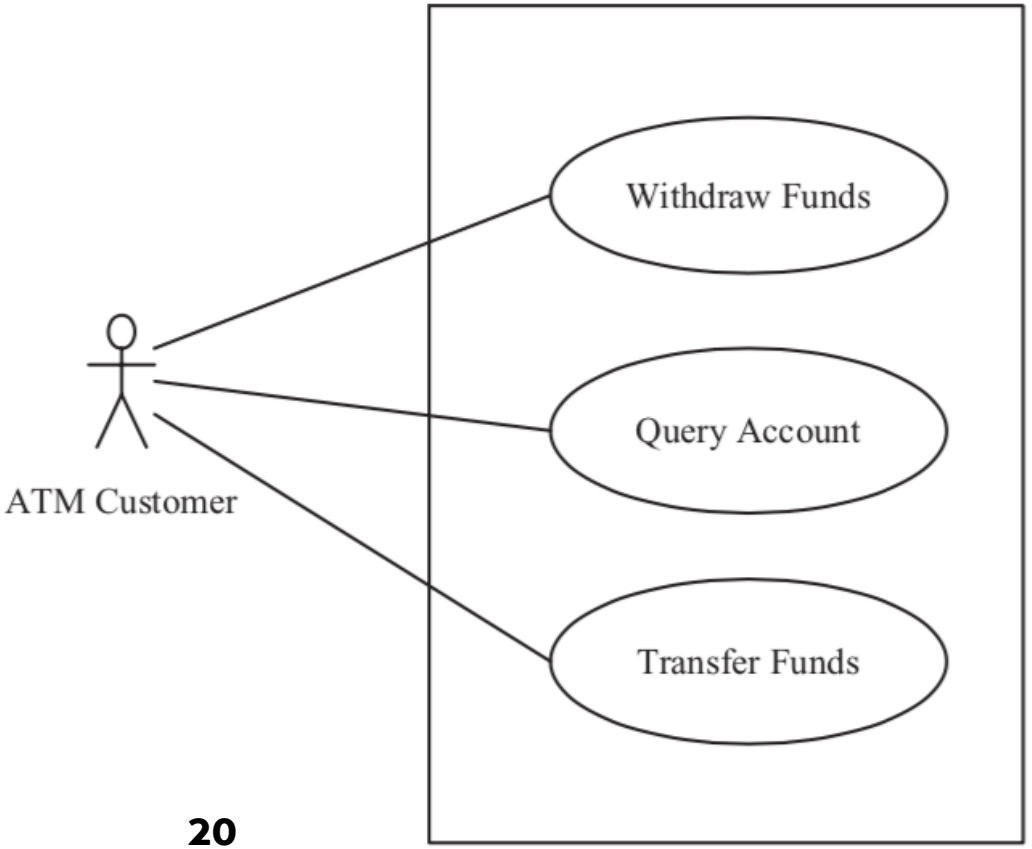
مثال 2: مخطط حالات الاستخدام: نظام مصرفي

الممثل:(Actor)

- **العميل (Customer):** المستخدم الذي يتفاعل مع النظام المصرفي.

حالات الاستخدام:(Use Cases)

الحالة	الوصف
سحب الأموال Withdraw Funds	يقوم العميل بطلب سحب مبلغ مالي من حسابه.
استعلام عن الحساب Query Account	يطلب العميل عرض رصيد الحساب أو تفاصيل المعاملات.
تحويل الأموال Transfer Funds	يُجري العميل تحويلًا ماليًا إلى حساب آخر داخل أو خارج البنك.



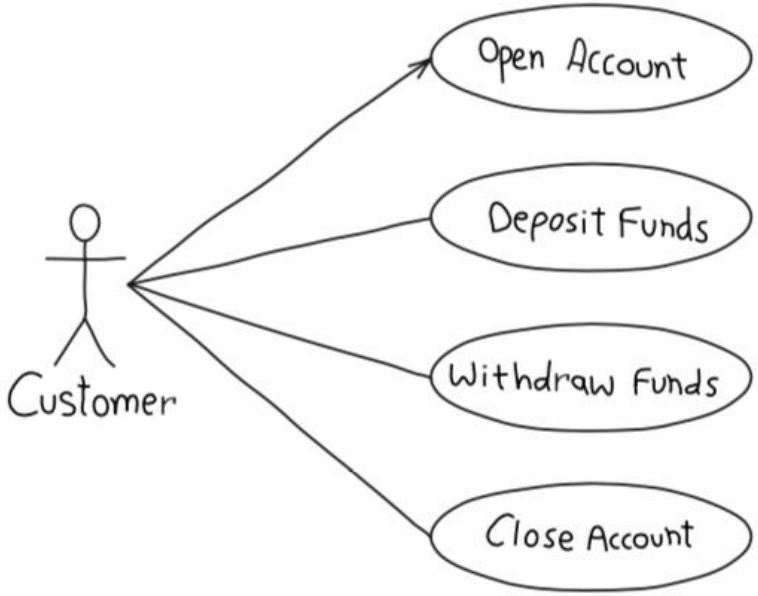
Banking System actor and use cases

USE-CASE GUIDELINES

- Begin Use-Case Names with a Strong Verb
- Name Use Cases Using Domain Terminology
- Imply Timing Considerations by Stacking Use Cases

Withdraw Funds ✓

Process Withdrawal Transaction ✗



إرشادات تسمية حالات الاستخدام في UML

. **ابدأ اسم حالة الاستخدام بفعل قوي**يُفضّل أن تبدأ أسماء حالات الاستخدام **بفعل يُعبّر عن الإجراء المطلوب**، مثل:

- ❖ "سحب الأموال"
- ❖ "عرض الرصيد"
- ❖ "تحديث الملف الشخصي"

استخدام الأفعال القوية يُوضّح الغرض من الحالة ويُسهّل فهمها.

. **استخدم مصطلحات المجال**

يُوصى باستخدام **مصطلحات مألوفة من مجال النظام** عند تسمية الحالات، مثل "طلب تحويل"، "إصدار فاتورة"، أو "تسجيل دخول"، مما يُعزّز الربط بين النموذج والواقع العملي.

الإيحاء بالترتيب الزمني من خلال التكديس عند رسم المخطط، يُمكن **ترتيب حالات الاستخدام بشكل رأسي** للإيحاء بالتسلسل الزمني أو الأولوية، مثل:

- في الأعلى: "تسجيل الدخول"
- في الوسط: "استعراض الحساب"
- في الأسفل: "إجراء تحويل"

هذا الترتيب يُساعد القارئ على فهم تدفق العمليات داخل النظام بشكل

IDENTIFYING ACTORS

An actor characterizes an external user (i.e., outside the system) that interacts with the system. in other words, actors are outside the system and not part of it.

An actor is a person, organization, local process (e.g., system clock), or external system that plays a role in one or more interactions with your system. It is very often a human user. For this reason, in UML, an actor is depicted using a ***stick figure***.

It is possible for an actor to be an external system that interfaces to the system, in some applications, an actor can also be an external I/O device or a timer(in real-time embedded systems).

في لغة النمذجة الموحدة UML ، يُمثّل **الممثل** Actor كيانًا خارجيًا يتفاعل مع النظام، ويُعد جزءًا أساسيًا في مخططات حالات الاستخدام.

خصائص الممثل:

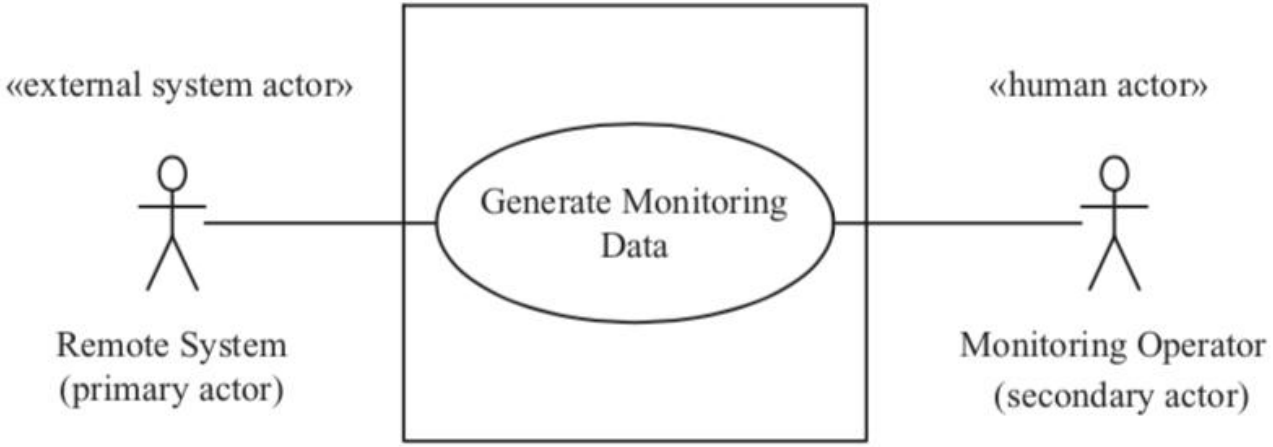
- يُمثّل **مستخدمًا خارجيًا** للنظام، أي أنه **ليس جزءًا من النظام نفسه**.
- يمكن أن يكون:
  - **شخصًا** (مثل مستخدم بشري)
  - **منظمة** (مثل جهة خارجية تتعامل مع النظام)
  - **عملية محلية** (مثل ساعة النظام)
  - **نظام خارجي** (مثل بوابة دفع إلكترونية)
  - **جهاز إدخال/إخراج خارجي** أو **مؤقت** (في الأنظمة المدمجة ذات الزمن الحقيقي)

يُستخدم الممثل لتحديد من يتفاعل مع النظام وما الدور الذي يلعبه في حالات الاستخدام المختلفة، مما يُساعد في فهم المتطلبات الوظيفية من وجهة نظر المستخدمين أو الأنظمة المتصلة.

IDENTIFYING ACTORS

❑ Primary and Secondary Actors

A **primary actor** initiates a use case. Thus, the use case starts with an input from the primary actor to which the system has to respond. Other actors, referred to as **secondary actors**, can participate in the use case.



Example of primary and secondary actors, as well as external system actor

في مخططات حالات الاستخدام ضمن لغة UML، يُصنّف الممثلون إلى نوعين رئيسيين:

الممثل الأساسي Primary Actor

- هو **الجهة التي تُبادر بتنفيذ حالة الاستخدام**.
- يُقدّم **المدخل الأول** للنظام، مما يُحفّز النظام على الاستجابة وتنفيذ سلسلة من التفاعلات.
- غالبًا ما يكون **المستخدم النهائي** أو جهة خارجية تطلب خدمة معينة من النظام.

مثال: في نظام مصرفي، العميل الذي يطلب "سحب الأموال" يُعد ممثلًا أساسيًا.

الممثل الثانوي Secondary Actor

هو **الجهة التي تُشارك في حالة الاستخدام** لكنها **لا تُبادر بها**.

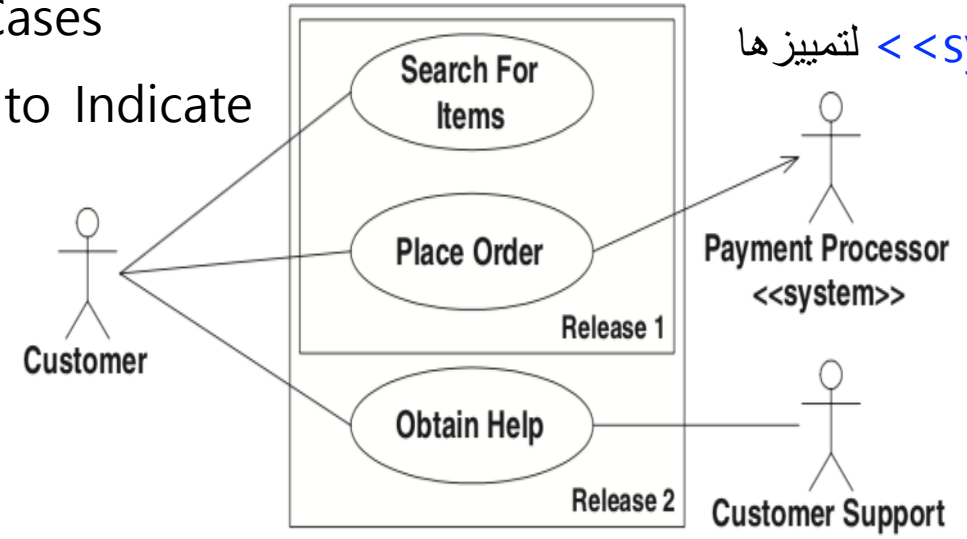
- يُقدّم خدمات أو معلومات داعمة للنظام أثناء تنفيذ الحالة.
- قد يكون نظامًا خارجيًا، قاعدة بيانات، أو جهازًا تقنيًا.

مثال: في نفس النظام المصرفي، قاعدة البيانات التي تُوفّر رصيد الحساب تُعد ممثلًا ثانويًا. هذا التصنيف يُساعد في فهم الأدوار المختلفة التي تلعبها الكيانات الخارجية أثناء تنفيذ الوظائف داخل النظام، ويُعزّز من دقة تحليل المتطلبات.

ACTOR GUIDELINES

- Place Your Primary Actor(s) in the Top Left Corner of the Diagram.
- Draw Actors on the Outside Edges of a Use-Case Diagram
- Name Actors with Singular, Domain-Relevant Nouns
- Associate Each Actor with One or More Use Cases
- Use <<system>> to Indicate System Actors

- **ضع الممثل الأساسي في الزاوية العلوية اليسرى** يُفضّل وضع الممثل أو الممثلين الأساسيين في **الزاوية العلوية اليسرى** من المخطط، لأنهم يُمثّلون نقطة البداية لحالات الاستخدام، مما يُسهّل تتبّع تدفق التفاعل.
- **ارسم الممثلين على الحواف الخارجية للمخطط** يجب وضع جميع الممثلين **خارج حدود النظام**، على أطراف المخطط، لأنهم كيانات خارجية تتفاعل مع النظام ولا يُعتبرون جزءًا منه.
- **سمّ الممثلين بأسماء مفردة وذات صلة بالمجال** استخدم **أسماء مفردة** تعكس المصطلحات الشائعة في مجال النظام، مثل: "عميل" بدلًا من "العملاء" "طالب" بدلًا من "الطلاب" "نظام دفع" بدلًا من "أنظمة الدفع"
- **اربط كل ممثل بحالة استخدام واحدة على الأقل** يجب أن يكون لكل ممثل **دور واضح** في المخطط، من خلال ارتباطه بحالة استخدام واحدة على الأقل، مما يُبرز علاقته بالنظام.
- **استخدم** <<system>> **للإشارة إلى الممثلين النظاميين** عند تمثيل **أنظمة خارجية** تتفاعل مع النظام، يُوصى باستخدام **الوسم** <<system>> لتمييزها عن المستخدمين البشريين، مثل: بوابة الدفع او نظام التحقق

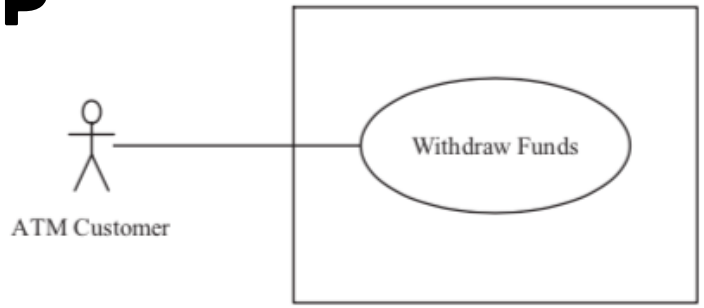


Online shopping.

IDENTIFYING USE CASE RELATIONSHIP

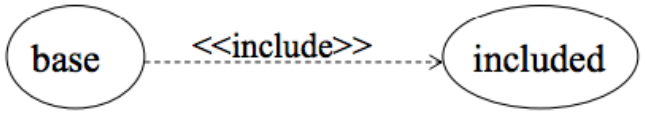
❖ Association Relationship

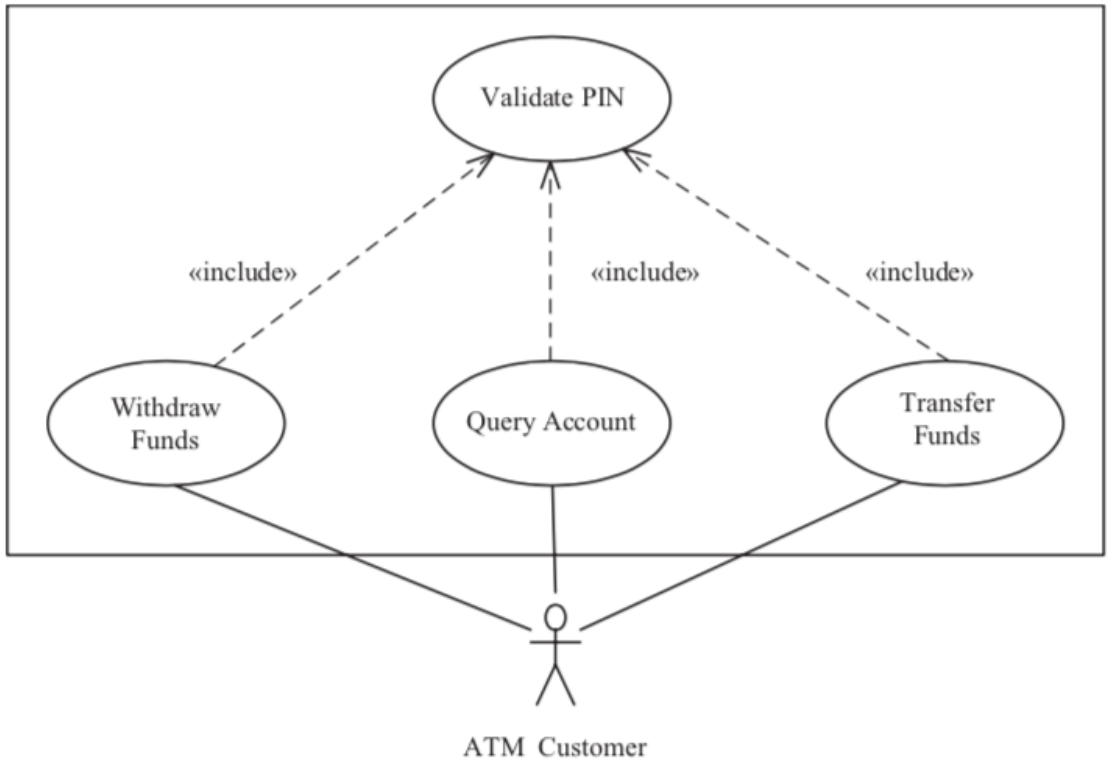
The Association Relationship represents a communication or interaction between an actor and a use case.



❖ Include Relationship

The Include Relationship indicates that a use case includes the functionality of another use case.

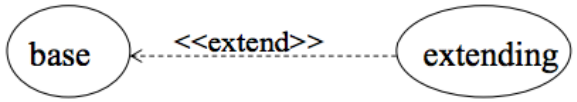




IDENTIFYING USE CASE RELATIONSHIP

❖ Extend Relationship

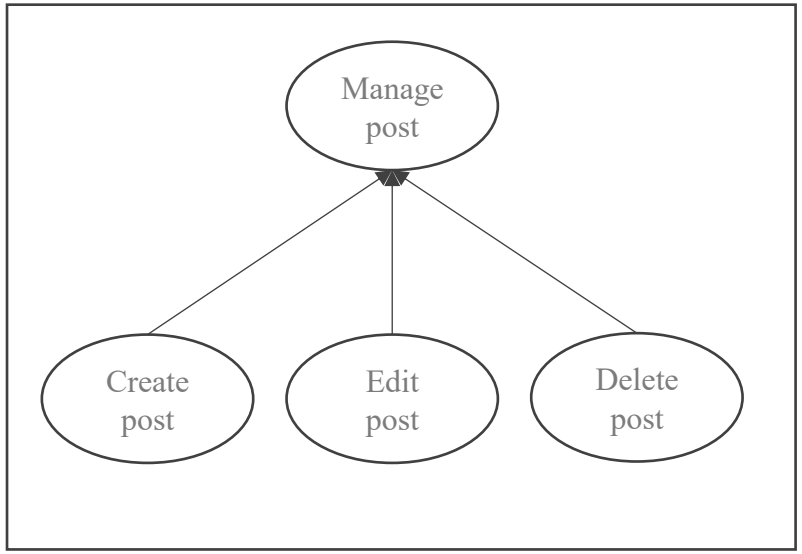
The Extend Relationship illustrates that a use case can be extended by another use case under specific conditions.



❖ Generalization Relationship

The Generalization Relationship establishes an "is-a" connection between two use cases, or two actors.

- 2 use case indicating that one use case is a specialized version of another.
- 2 actors represents a hierarchical relationship where one actor is a more specialized version of another actor.



IDENTIFYING USE CASE RELATIONSHIP

1. علاقة الارتباط Association Relationship

تُستخدم لربط **ممثل** Actor بحالة استخدام.

تُعبّر عن وجود **تفاعل مباشر** بين الممثل والنظام.

تُرسم بخط مستقيم بين الممثل وحالة الاستخدام.

مثال: "العميل" مرتبط بحالة "سحب الأموال".

2. علاقة التضمين Include Relationship

تُستخدم عندما **تتضمّن حالة استخدام أخرى بشكل دائم**.

تُشير إلى أن الحالة المُضمّنة تُنفّذ دائمًا كجزء من الحالة الأصلية.

تُرسم بسهم متقطع مع وسم include

مثال: "تحويل الأموال" تتضمّن دائمًا "التحقق من الرصيد".

3. علاقة الامتداد Extend Relationship

تُستخدم عندما **تُنفّذ حالة استخدام إضافية بشكل اختياري**.

تُشير إلى أن الحالة الممتدة تُنفّذ فقط في ظروف معينة.

تُرسم بسهم متقطع مع وسم extend

مثال: "سحب الأموال" قد تمتد إلى "إرسال إشعار بالبريد الإلكتروني" إذا تم تفعيل التنبيهات.

4. علاقة التعميم Generalization Relationship

تُستخدم عندما **يرث ممثل أو حالة استخدام خصائص من أخرى**.

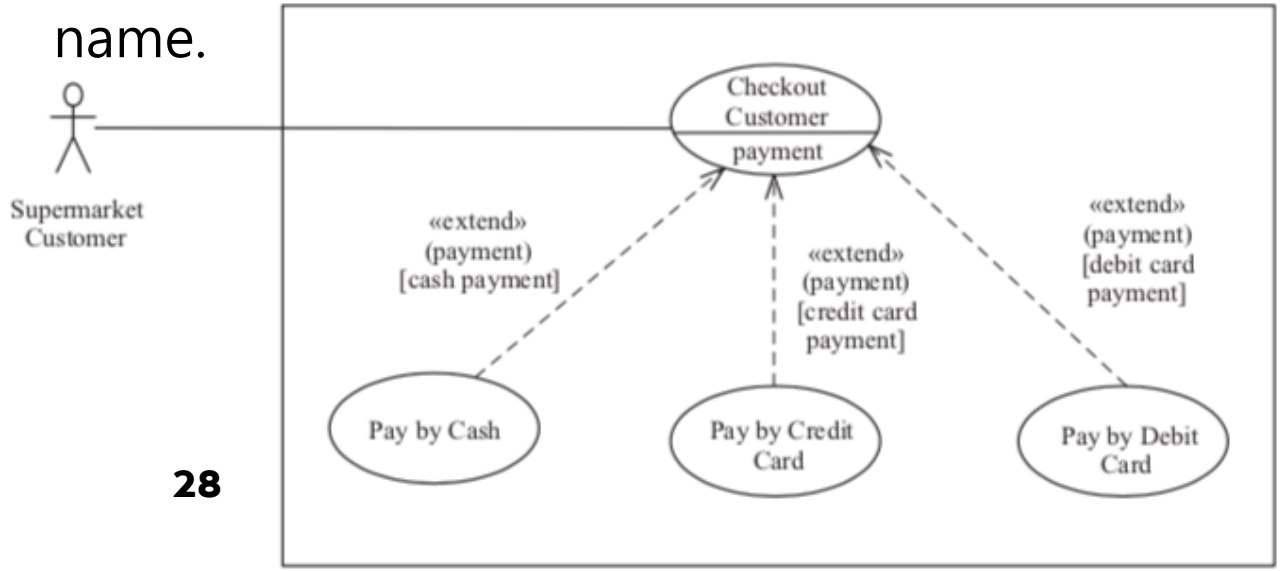
تُعبّر عن علاقة "هو نوع من" is-a

تُرسم بسهم مفتوح الرأس من العنصر الفرعي إلى العنصر الأساسي.

مثال: "مستخدم مميز" يُعد تعميمًا من "مستخدم عادي".

EXTENSION POINTS

- Extension points are used to specify the precise locations in the base use case at which extensions can be added. An extension use case may extend the base use case only at these extension points.
- Each extension point is given a name.



نقاط الامتداد في UML (Extension Points)

نقاط الامتداد هي مواقع محددة داخل **حالة استخدام أساسية** Base Use Case يُمكن عندها إدراج **حالات استخدام ممتدة** Extension Use Cases

- تُحدّد بدقة **مكان إدراج التوسعة** داخل تسلسل التفاعل الأساسي.
- تُستخدم فقط عندما تكون هناك حاجة لإضافة سلوك اختياري أو شرطي إلى الحالة الأساسية.
- كل نقطة امتداد يجب أن تُعطى **اسمًا فريدًا** يُعبّر عن موقعها أو وظيفتها داخل الحالة الأساسية.

تُنشأ حالة استخدام ممتدة (مثل "التحقق من الهوية") وتُربط بالحالة الأساسية (مثل "سحب الأموال") عبر علاقة `extend`

- يُحدّد في الحالة الأساسية نقطة امتداد باسم مثل: التحقق الأمني.
- يُمكن تنفيذ الحالة الممتدة فقط عند تلك النقطة، وفي ظروف معينة (مثل تجاوز مبلغ معين).

هذا الأسلوب يُساعد على **تنظيم السلوك الشرطي** داخل النظام دون تعقيد الحالة الأساسية، ويُعزّز من قابلية التوسع وإعادة الاستخدام.

RELATIONSHIP GUIDELINES

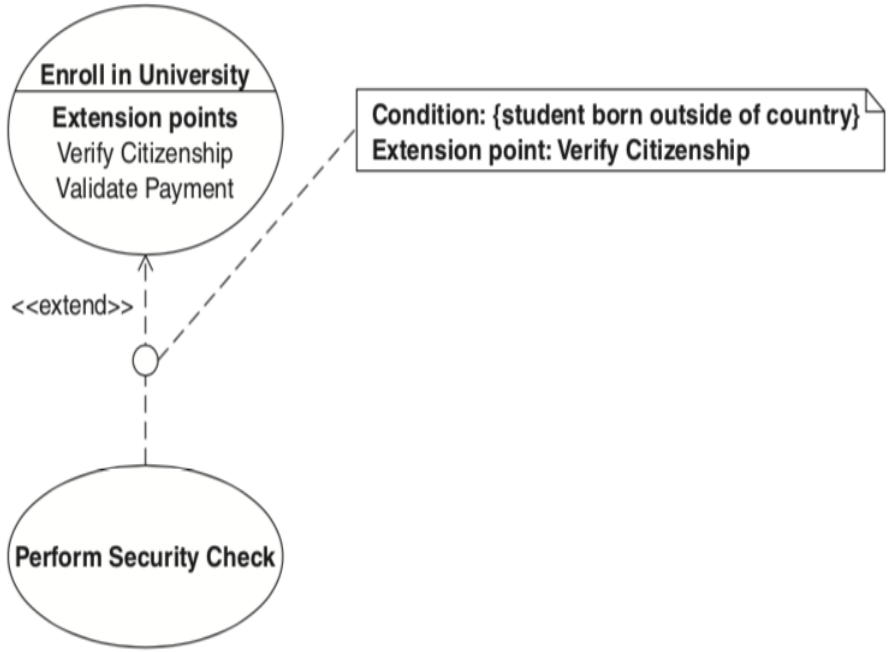
- Avoid Arrowheads on Actor–Use-Case Relationships
- Do Not Apply <<uses>>, <<includes>>, or <<extends>>
- Place an Included Use Case to the Right of the Invoking Use Case
- Place an Extending Use Case Below the Parent Use Case

إرشادات متقدمة لتصميم مخططات حالات الاستخدام في UML

- **تجنّب رؤوس الأسهم بين الممثلين وحالات الاستخدام** يُفضّل استخدام **خطوط بدون رؤوس أسهم** عند الربط بين الممثل Actor وحالة الاستخدام، لأن العلاقة تُعبّر عن تفاعل وليس عن اتجاه تنفيذ.
- **لا تستخدم** uses**أو** includes**أو** extends يُوصى بتجنّب استخدام هذه الوسوم في المخطط البصري، لأنها قد تُربك القارئ أو تُعقّد التصميم. يُمكن توضيح العلاقات من خلال الترتيب والموقع بدلًا من الوسوم النصية.
- **ضع حالة الاستخدام المُضمّنة إلى يمين الحالة المُستدعِية** عند وجود علاقة تضمين Include ، يُفضّل وضع الحالة المُضمّنة **على يمين** الحالة التي تستدعيها، مما يُوضّح التسلسل المنطقي بصريًا.
- **ضع حالة الاستخدام الممتدة أسفل الحالة الأصلية** في حالة وجود علاقة امتداد Extend، يُفضّل وضع الحالة الممتدة **أسفل** الحالة الأساسية، مما يُشير إلى أنها تُنفّذ بشكل اختياري أو شرطي.

RELATIONSHIP GUIDELINES

- Place an Inheriting Use Case Below the Base Use Case
- Avoid Modeling Extension Points
- Model Extension Conditions Only When They Aren’t Clear



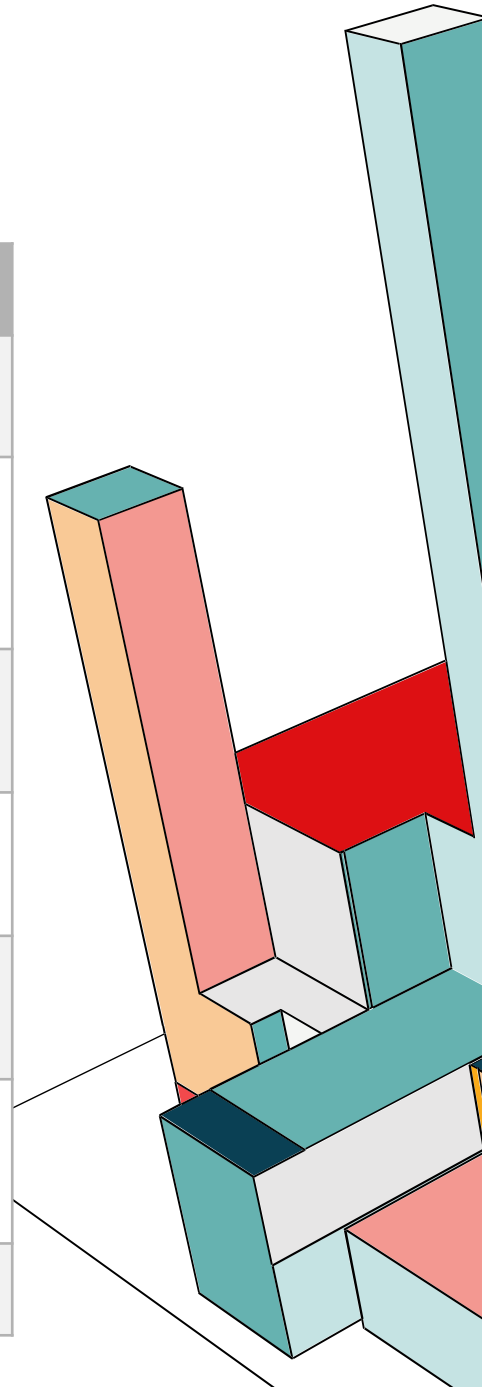
- **ضع حالة الاستخدام الوراثية أسفل الحالة الأساسية** عند وجود علاقة تعميم Generalization ، يُفضّل وضع الحالة الوراثية **أسفل** الحالة الأصلية، مما يُوضّح التسلسل الهرمي للعلاقات.
- **تجنّب نمذجة نقاط الامتداد** Extension Points يُوصى بعدم تضمين نقاط الامتداد داخل المخطط، لأنها تُضيف تعقيدًا غير ضروري، ويمكن توضيح السلوك الشرطي بطرق أبسط.
- **نمذجة شروط الامتداد فقط عند عدم وضوحها** لا حاجة لتوضيح شروط الامتداد إلا إذا كانت **غير واضحة أو قد تُفسّر بشكل خاطئ**. في هذه الحالة، يُمكن استخدام ملاحظة نصية لتوضيح الشرط.

هذه الإرشادات تُساعد على تبسيط المخطط، وتحسين قابليته للقراءة، وتُقلّل من التشويش البصري، خصوصًا عند مشاركة المخططات مع فرق متعددة التخصصات.

DOCUMENTING USE CASES IN THE USE CASE MODEL

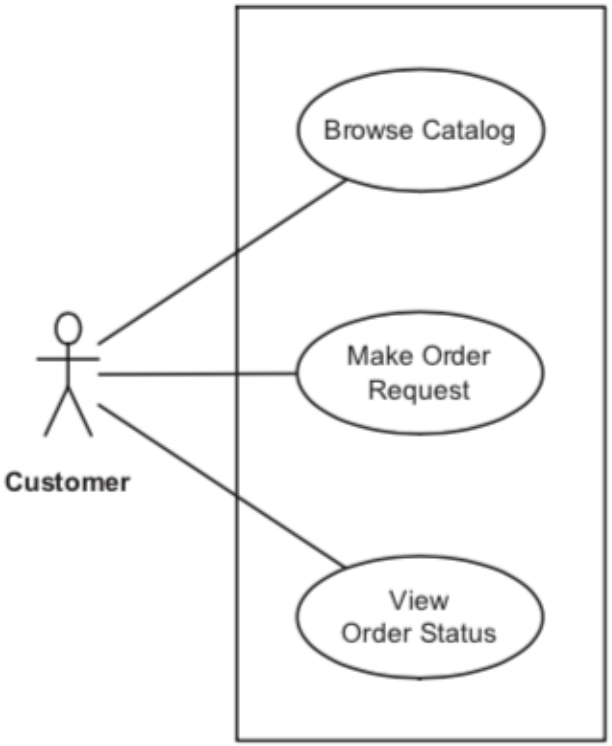
Each use case in the use case model is documented in a use case description, as follows:

Use case name:	Each use case is given a name.
Summary	A brief description of the use case, typically one or two sentences.
Dependency	This optional section describes whether the use case depends on other use cases – that is, whether it includes or extends another use case.
Actors	This section names the actors in the use case. There is always a primary actor that initiates the use case.
Preconditions	One or more conditions that must be true at the start of use case, from the perspective of this use case
Description of main sequence	description of the main sequence of the use case, which is the most usual sequence of interactions between the actor and the system.
Description of alternative sequences	Narrative description of alternative branches off the main sequence.
Postcondition	Condition that is always true at the end of the use case



DOCUMENTING USE CASES IN THE USE CASE MODEL

Example Of Use Case Description:



Use case name:	Make Order Request
Summary (Description)	Customer enters an order request to purchase items from the online shopping system. The customer's credit card is checked for sufficient credit to pay for the requested catalog items.
Actors	Customer
Preconditions	The customer has selected one or more catalog items.
Description of main sequence	1. Customer provides order request and customer account Id to pay for purchase. 2. System retrieves customer account information, including the customer's credit card details. 3. System checks the customer's credit card for the purchase amount and, if approved, creates a credit card purchase authorization number. 4. System creates a delivery order containing order details, customer Id, and credit card authorization number. 5. System confirms approval of purchase and displays order information to customer.
Description of Alternative sequences:	Step 2: If customer does not have account, the system creates an account. Step 3: If the customer's credit card request is denied, the system prompts the customer to enter a different credit card number. The customer can either enter a different credit card number or cancel the order.
Postcondition	1. System has created a delivery order for the customer.

USE CASE PACKAGES

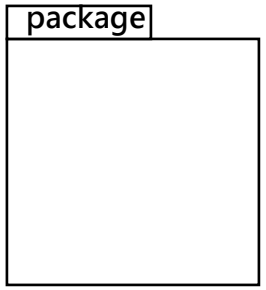
في الأنظمة الكبيرة، قد يحتوي نموذج حالات الاستخدام على **عدد كبير من الحالات**، مما يجعل إدارة النموذج وتصفّحه أمرًا صعبًا ومعقّدًا. للتعامل مع هذا التوسّع، يُوصى باستخدام **حزم** Packages لتجميع حالات الاستخدام ذات الصلة ضمن مجموعات منطقية. هذه الحزم تُسهم في:

- **تنظيم النموذج** بطريقة تُسهّل الفهم والصيانة.
- **تقسيم النظام إلى وحدات وظيفية** أصغر وأكثر قابلية للإدارة.
- **تحديد نطاق كل مجموعة** من حالات الاستخدام بوضوح.

أمثلة على الحزم:

- ❑ في نظام مصرفي: حزمة "الخدمات المالية": تشمل "سحب الأموال"، "تحويل الأموال"، "إيداع". حزمة "إدارة الحساب": تشمل "تحديث البيانات"، "عرض الرصيد"، "إغلاق الحساب".
- ❑ في نظام تعليمي: حزمة "إدارة الطلاب": تشمل "تسجيل طالب"، "تعديل بيانات"، "عرض النتائج". حزمة "إدارة الدروس": تشمل "إضافة درس"، "تعديل محتوى"، "تحديد مواعيد".

Example of use case package



يُمكن تمثيل الحزم في UML باستخدام مستطيل يحتوي على اسم الحزمة، وتُوضع داخله حالات الاستخدام المرتبطة بها.

UML Class Diagram

- A class diagram is a static structure diagram in the Unified Modeling Language (UML) that represents the structure and behavior of a system or application through its classes, attributes, methods, and relationships.
- In class diagram, classes are depicted as boxes, and the static relationships between them are depicted as lines connecting the boxes.
- The class diagram describes the classes of applications being modeled along with their relationship.

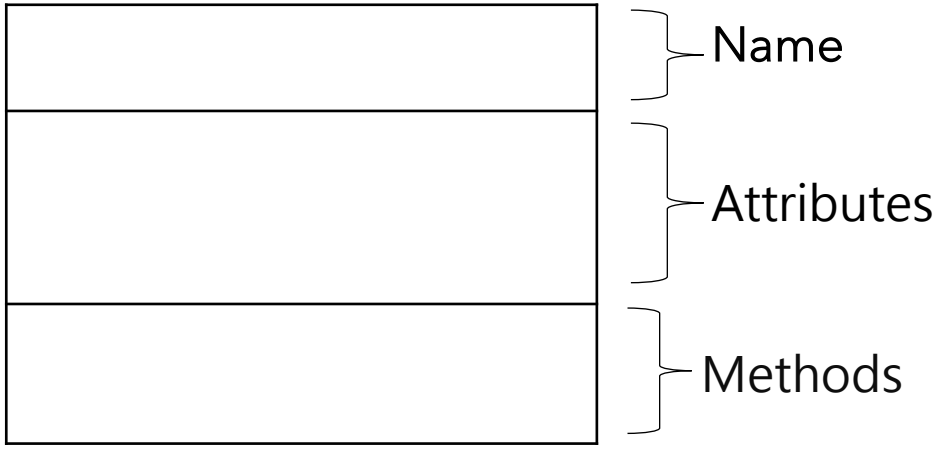
يعد **مخطط الفئات** Class Diagram أحد المخططات البنيوية الساكنة في لغة النمذجة الموحدة UML ، حيث يُستخدم لتمثيل البنية والسلوك الخاص بالنظام أو التطبيق من خلال الفئات Classes ، والسمات Attributes ، والعمليات أو الأساليب Methods ، إضافةً إلى العلاقات التي تربط بينها.

في هذا المخطط، تُعرض الفئات على شكل مربعات، بينما تُظهر الخطوط الواصلة بين هذه المربعات العلاقات الساكنة التي تربط الفئات ببعضها البعض. ويُسهم مخطط الفئات في وصف الفئات المكوِّنة للتطبيق محل النمذجة، إلى جانب توضيح طبيعة العلاقات التي تجمعها، مما يجعله أداة أساسية لفهم الهيكل العام للنظام وتوثيقه.

Fundamental concepts of Class diagram

❖ Class

- Represents objects or Entities that share similar attributes, operations, relationships, and behaviors.
- can identify a class by a box of three compartment.
- Each class typically has a name and Define Attributes (variables) and Behaviors (methods).



تمثل **الفئة** Class مجموعة من الكائنات أو الكيانات التي تشترك في سمات، وعمليات، وعلاقات، وسلوكيات متشابهة.

يمكن تمييز الفئة في مخطط الفئات من خلال صندوق يتكون عادةً من ثلاثة أقسام رئيسية. يحتوي القسم الأول على اسم الفئة، بينما يضم القسم الثاني السمات Attributes التي تُعرّف المتغيرات المرتبطة بها، ويعرض القسم الثالث السلوكيات Methods التي تمثل العمليات أو الوظائف التي يمكن للفئة تنفيذها.

بهذا الشكل، يُعد مخطط الفئات أداة أساسية لتوضيح البنية الداخلية للنظام، حيث يتيح فهم كيفية تنظيم البيانات والوظائف داخل كل فئة، وكذلك طبيعة العلاقات التي تربطها بالفئات الأخرى.

Fundamental concepts of Class diagram

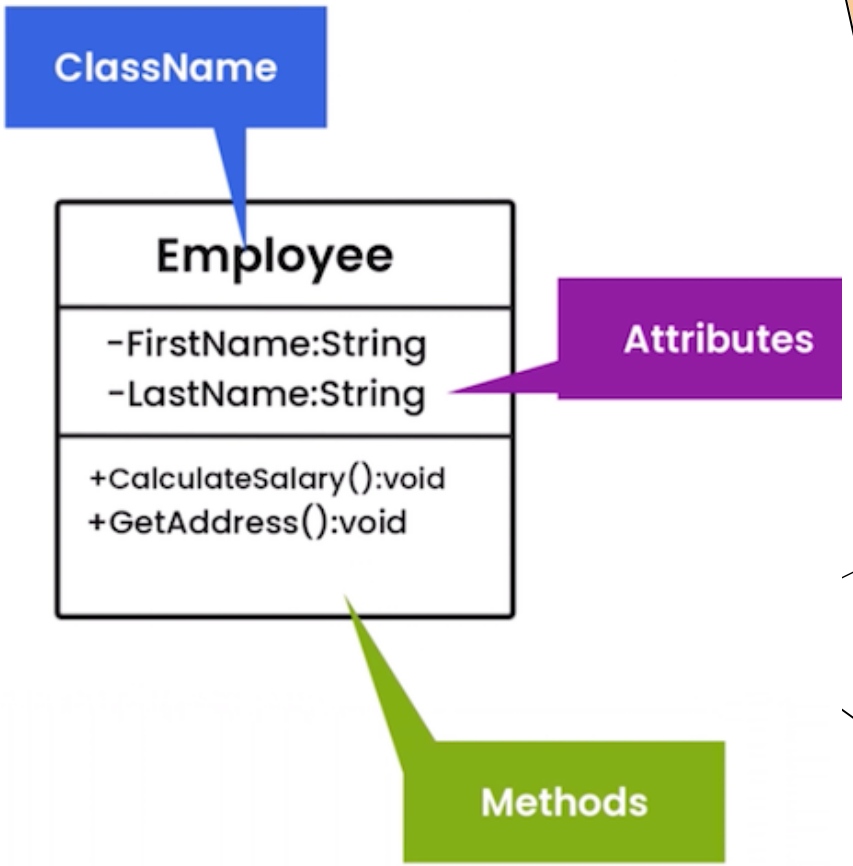
- تمثل السمات الخصائص أو المميزات المرتبطة بالفئة، وهي التي تصف الحالة الداخلية للكائن Object المنتمي إلى تلك الفئة. عادةً ما تُعرض هذه السمات على شكل متغيرات داخل الفئة، وتُستخدم لتحديد القيم التي تميز كل كائن عن الآخر.
- بينما تشير الأساليب إلى الأفعال أو العمليات التي يمكن للفئة تنفيذها. وهي تعكس السلوكيات أو الوظائف التي يقوم بها الكائن، وتُعرّف عادةً على شكل إجراءات أو دوال داخل الفئة، مما يتيح التفاعل مع البيانات وتغيير حالتها.

Attributes:

Properties or characteristics of class. They describe the state of an object and are typically shown as variables within the class.

Methods:

Actions that a class can perform.



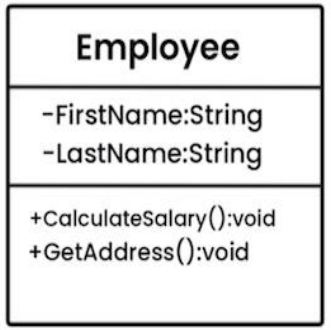
From Diagram to Code.

Class diagram represents the blueprint of the code.

مخطط الفئات Class Diagram بمثابة المخطط الأساسي أو "المخطط الهندسي" Blueprint للشفرة البرمجية، إذ يوفّر تصوراً بنيوياً يوضح كيفية تنظيم الفئات، وسماتها، وأساليبها، والعلاقات التي تربطها ببعضها البعض داخل النظام. ومن خلال هذا المخطط يمكن للمطورين والمحللين فهم البنية العامة للتطبيق قبل البدء في عملية التنفيذ البرمجي الفعلي.

على سبيل المثال، يمكن تمثيل **فئة موظف** Employee Class في مخطط الفئات كما يلي:

- **اسم الفئة:** Employee
- **السمات** Attributes :
  - firstName : String
  - lastName : String
- **الأساليب** Methods :
  - calculateSalary() : Double
  - getAddress() : String



Employee.cs

```
1   public class Employee
2   {
3       private string FirstName { get; set; }
4       private string LastName { get; set; }
5
6        public void CalculateSalary()
7        {
8
9        }
10       public void GetAddress()
11       {
12
13       }
14
15  }
16
```

هذا المثال يوضح أن فئة "الموظف" تحتوي على سمات أساسية مثل الاسم الأول والاسم الثاني، بالإضافة إلى أساليب (وظائف) مثل حساب المرتب ومعرفة عنوان السكن. وبذلك يصبح المخطط مرجعاً بصرياً يوضح البنية الداخلية للفئة قبل تحويلها إلى شفرة برمجية فعلية.

Fundamental concepts of Class diagram

Visibilities

Define, which classes in the diagram have access to certain variables and methods.

This concept is used in object-oriented programming.

There are four visibilities types:

public	+	Accessible anywhere and within System
private	–	Accessible in a class that defines it.
protected	#	Accessible in a class that defines it OR Subclass
package	~	Accessible within same package

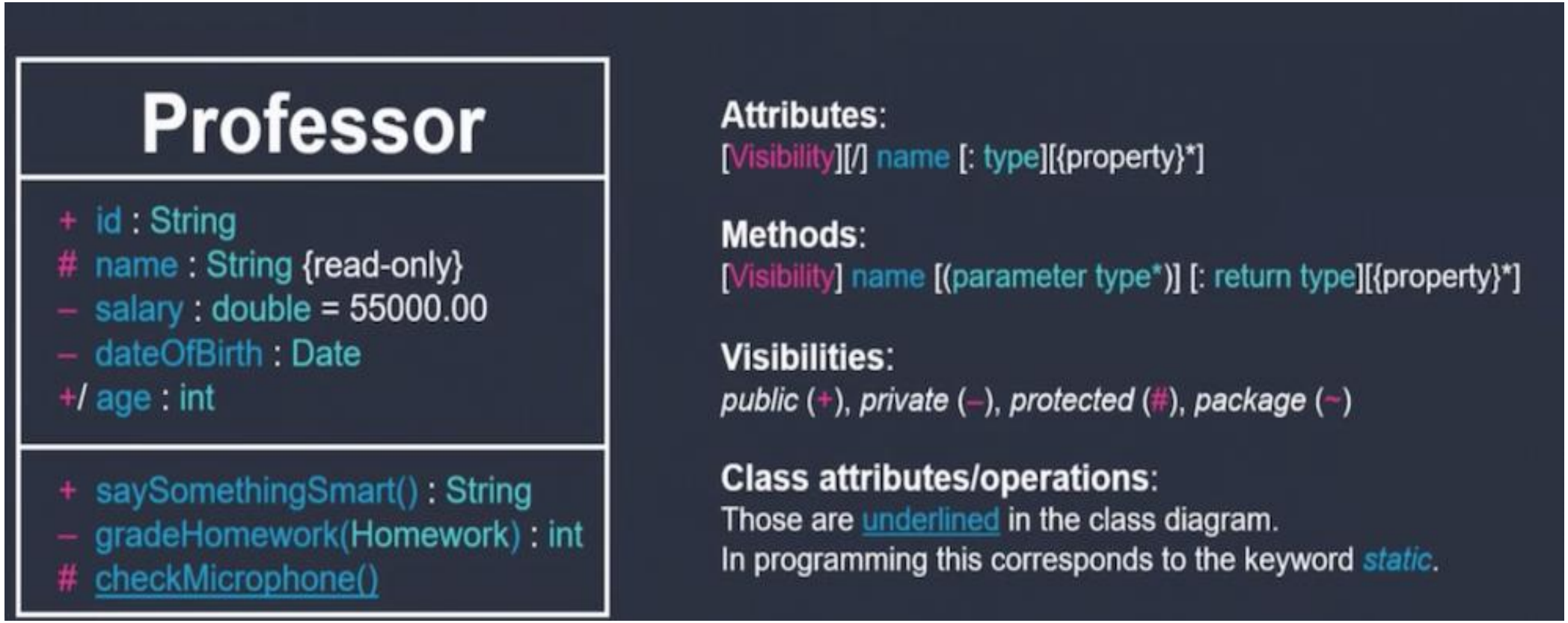
في البرمجة الكائنية التوجه هناك مفهوم مهم يُسمى **الرؤية** Visibility أو **مستوى الوصول** Access Modifiers ، وهو الذي يحدد أي الفئات Classes أو الكائنات يمكنها الوصول إلى المتغيرات Variables أو الأساليب Methods داخل الفئة.

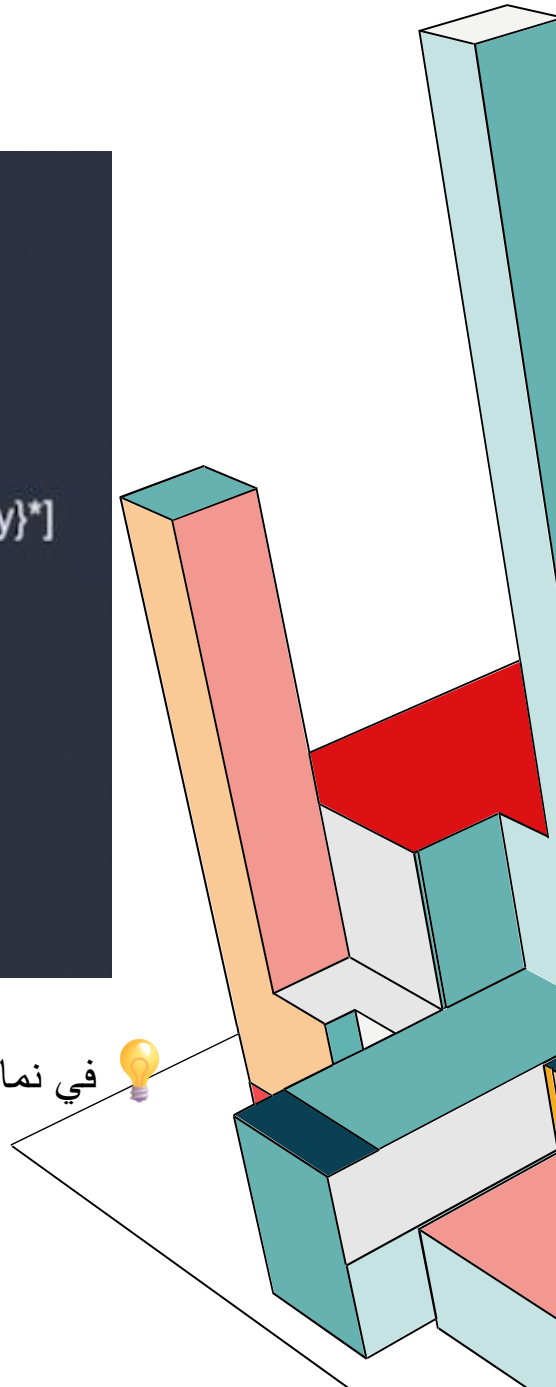
هذا المفهوم يُستخدم أيضاً في **مخطط الفئات** Class Diagram ضمن لغة UML لتوضيح حدود الوصول.

الأنواع الأربعة للرؤية Visibilities Types

1. Public **عام :** يمكن الوصول إلى المتغير أو الأسلوب من أي فئة أخرى داخل النظام. يُرمز له عادةً بعلامة **+** في مخطط الفئات.
2. Private **خاص :** لا يمكن الوصول إلى المتغير أو الأسلوب إلا من داخل نفس الفئة. يُرمز له بعلامة **-**.
3. Protected **محمي :** يمكن الوصول إليه من داخل الفئة نفسها أو من الفئات المشتقة Subclasses. يُرمز له بعلامة **#**.
4. Package **افتراضي/مستوى الحزمة :** يمكن الوصول إليه فقط من الفئات الموجودة داخل نفس الحزمة Package. يُرمز له بعلامة **~**.

Fundamental concepts of Class diagram





💡 في نماذج التصميم التفصيلية ينبغي دائماً تحديد مستوى الرؤية Visibility لكل من السمات Attributes والعمليات Operations .
فإظهار نوع الوصول (عام، خاص، محمي، أو مستوى الحزمة) يساهم في توضيح كيفية تعامل الفئات مع بياناتها وأساليبها،
ويعزز من دقة النموذج ووضوحه أثناء عملية التحليل والتصميم.

Fundamental concepts of Class diagram

❖ Relations

1) Associations

2) Aggregation

3) Composition

4) Inheritance

1. Association **الارتباط :** علاقة عامة تربط بين فئتين، مثل علاقة "طالب" ↔ "مقرر دراسي".

2. Aggregation **التجميع :** علاقة "الكل-الجزء" حيث يمكن للجزء أن يوجد بشكل مستقل عن الكل. مثال: "قسم" يحتوي على "موظفين"، لكن الموظف يمكن أن يوجد خارج القسم.

3. Composition **التركيب :** علاقة قوية من نوع "الكل-الجزء" حيث لا يمكن للجزء أن يوجد مستقلاً عن الكل. مثال: "منزل" يحتوي على "غرف"، فإذا أُزيل المنزل تُزال الغرف معه.

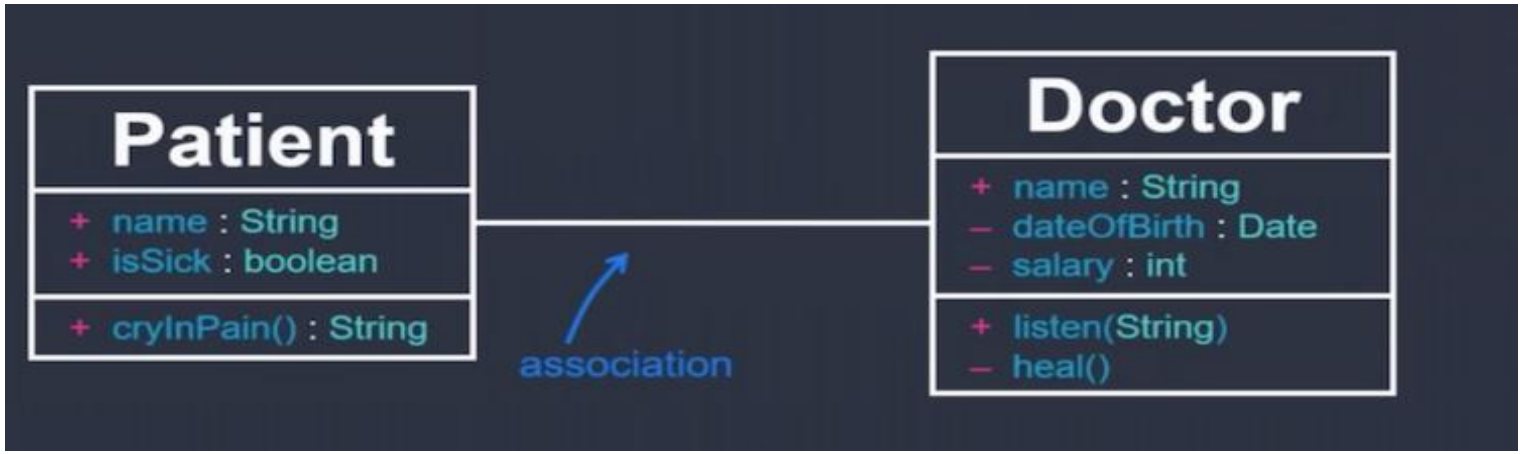
4. Inheritance **الوراثة/التعميم :** علاقة تُظهر أن فئة ما ترث خصائص وسلوكيات من فئة أخرى. مثال: "طالب دراسات عليا" يرث من "طالب".

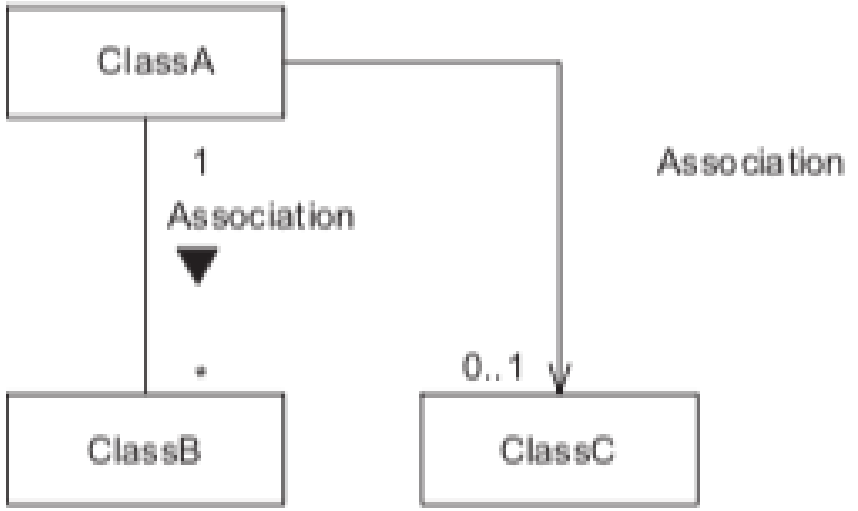
بهذا الشكل، يوضح مخطط الفئات ليس فقط البنية الداخلية للفئات، بل أيضاً طبيعة العلاقات التي تحدد كيفية تفاعلها وتكاملها داخل النظام.

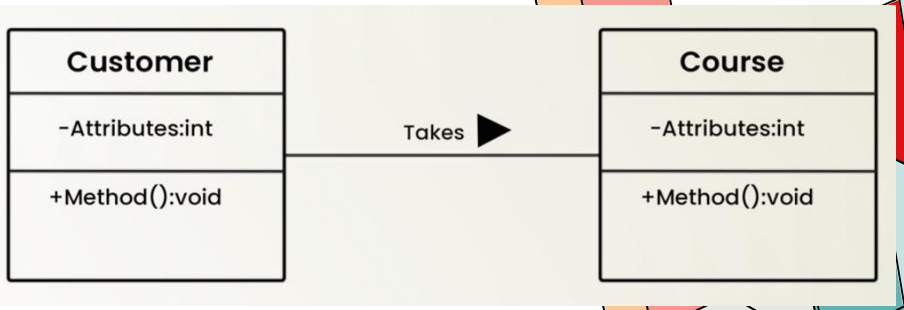
Relationships between Classes

❑ Associations:

How objects of one class interact with objects of another class







❖ Multiplicity

- For each class involved in a relationship, there will always be a multiplicity.

تشير **التعددية** في مخطط الفئات Class Diagram إلى عدد الكائنات Objects من فئة معينة التي يمكن أن ترتبط بكائن واحد من فئة أخرى ضمن علاقة محددة. وبذلك فهي تُستخدم لتوضيح طبيعة العلاقات بين الفئات من حيث الكم، أي عدد العناصر الممكنة في كل طرف من أطراف العلاقة.

لكل فئة مشاركة في علاقة ما، يجب دائماً تحديد قيمة التعددية الخاصة بها.

تُكتب التعددية عادةً بجانب خط العلاقة بين الفئات، مثل:

- **1** : يشير إلى أن هناك كائناً واحداً فقط يمكن أن يرتبط.
- **1..0** : يشير إلى أن العلاقة اختيارية (قد يوجد كائن واحد أو لا يوجد).
- **0..*** أو ***** : يشير إلى إمكانية وجود عدد غير محدد من الكائنات.
- **1..*** : يشير إلى وجود كائن واحد على الأقل، وقد يكون هناك أكثر.

Table 6. UML Multiplicity Indicators

Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
1..*	One or more
n	Only *n* (where *n* > 1)
*	Many
0..*n*	Zero to *n* (where n > 1)
1..*n*	One to *n* (where *n* > 1)
n..*m*	Where *n* and *m* both > 1
n..*	*n* or more, where *n* > 1

✨مثال توضيحي

في نظام مكتبة:

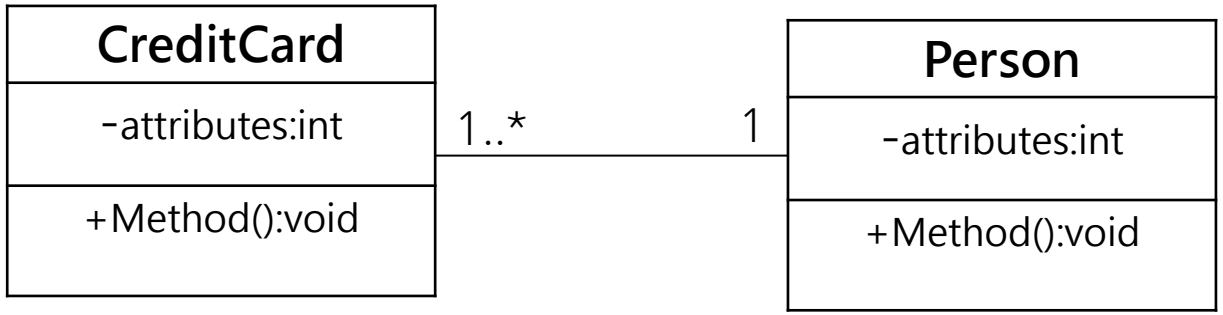
- فئة **طالب** Student ↔ فئة **مقرر دراسي** Course
- العلاقة: الطالب يمكن أن يسجل في عدة مقررات، والمقرر يمكن أن يحتوي على عدة طلاب.
- التعددية:
  - عند الطالب: **0..*** (يمكن أن يسجل في أي عدد من المقررات أو لا يسجل).
  - عند المقرر: **1..*** (يجب أن يحتوي على طالب واحد على الأقل).

Multiplicity

❖ Multiplicity Examples

- Many to Many(N:N)

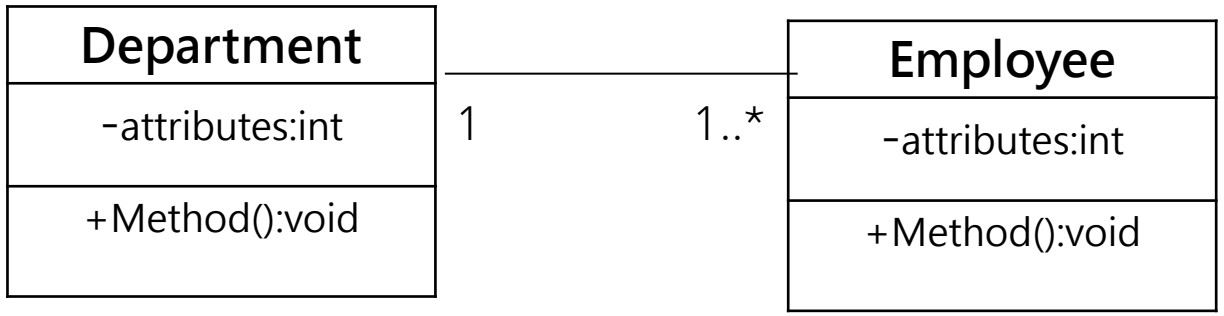
- Many to One(N:1)



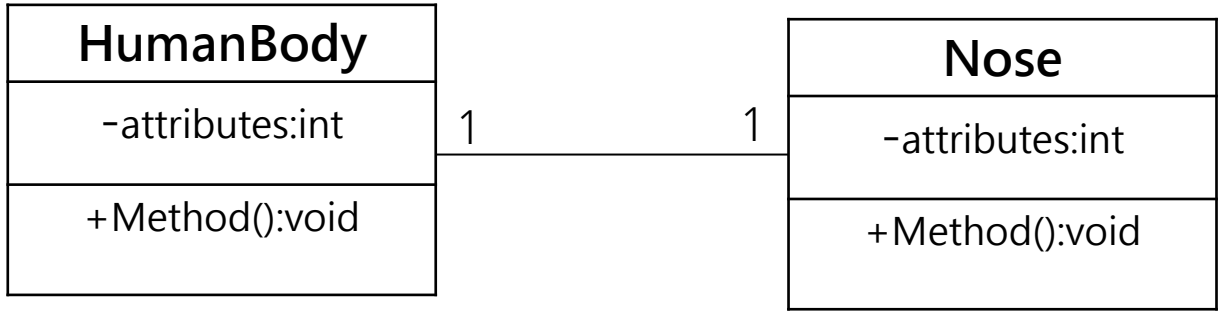
Multiplicity

❖ Multiplicity Examples

- One to Many(1:N)



- One to One(1:1)

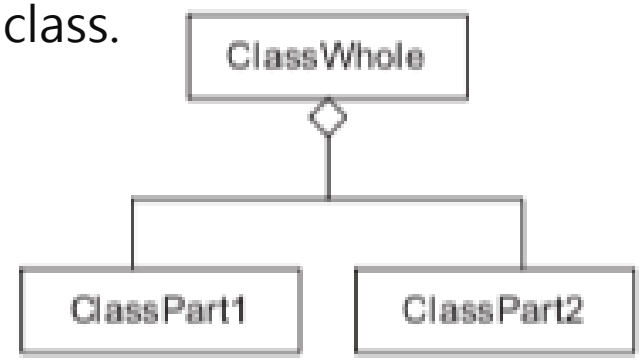


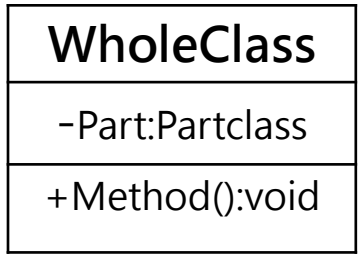
Relationships between Classes

❑ Aggregation:

Represents a strong whole-part relationship between two classes.

Partclass can exist Independently of the Whole class.



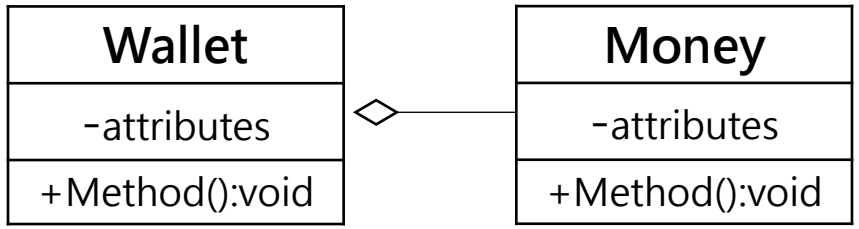


التجميع Aggregation

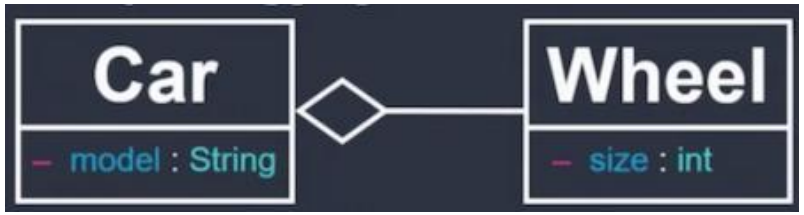
يمثل **التجميع** نوعاً من العلاقات القوية بين فئتين، يقوم على مبدأ "الكل-الجزء".

في هذا السياق، تُعتبر إحدى الفئات بمثابة الكل Whole Class ، بينما تُعد الفئة الأخرى جزءاً منه Part Class. وعلى الرغم من ارتباط الفئتين بعلاقة بنيوية، إلا أن الفئة الجزئية يمكن أن تستمر في الوجود بشكل مستقل عن الفئة الكلية.

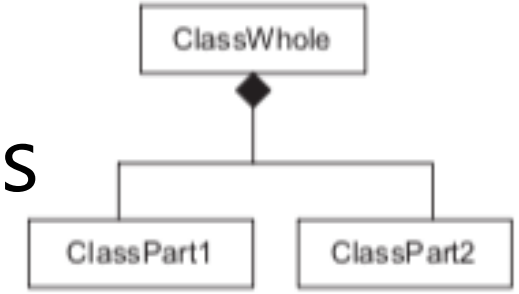
.



- مثال
- فئة **سيارة** ↔ **(Car)** فئة **عجلة القيادة (Wheel)**
- العلاقة: السيارة تحتوي على عجلة قيادة، لكن عجلة القيادة يمكن أن توجد بشكل مستقل (مثلاً كقطعة غيار في متجر أو مخزنة خارج السيارة).

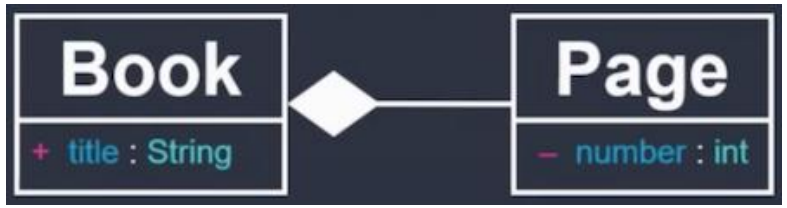


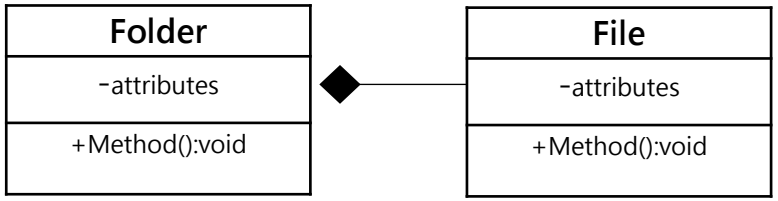
Relationships between Classes



❑ Composition:

Represents a strong whole-part relationship between two classes. Part cannot exist Independently of the Whole





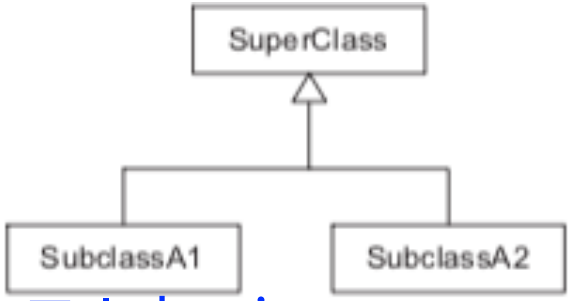
يمثل **التركيب** علاقة قوية من نوع "الكل-الجزء" بين فئتين، حيث يكون الجزء مرتبطاً ارتباطاً وجودياً بالكل، أي أنه لا يمكن أن يوجد بشكل مستقل عنه. فإذا تم حذف الكل، فإن الأجزاء المرتبطة به تُحذف تلقائياً. ويُستخدم هذا النوع من العلاقات في مخططات الفئات Class Diagrams للتعبير عن التبعية الكاملة بين الكائنات.

أمثلة توضيحية:

- **كتاب** Book ↔ **صفحة** Page: الصفحة لا يمكن أن توجد مستقلة عن الكتاب، فهي جزء أساسي من تكوينه، وإذا أُزيل الكتاب تُزال الصفحات معه.
- **مجلد** Folder ↔ **ملف** File: الملف في هذا السياق يُعتبر جزءاً من المجلد، ولا يمكن أن يوجد خارج المجلد الذي يحتويه. حذف المجلد يؤدي إلى حذف الملفات المرتبطة به.

الرمز ◆ (المعين المملوء) يُستخدم في UML لتمثيل علاقة **التركيب** Composition ، وهو يوضح أن الجزء لا يمكن أن يوجد مستقلاً عن الكل.

Relationships between Classes

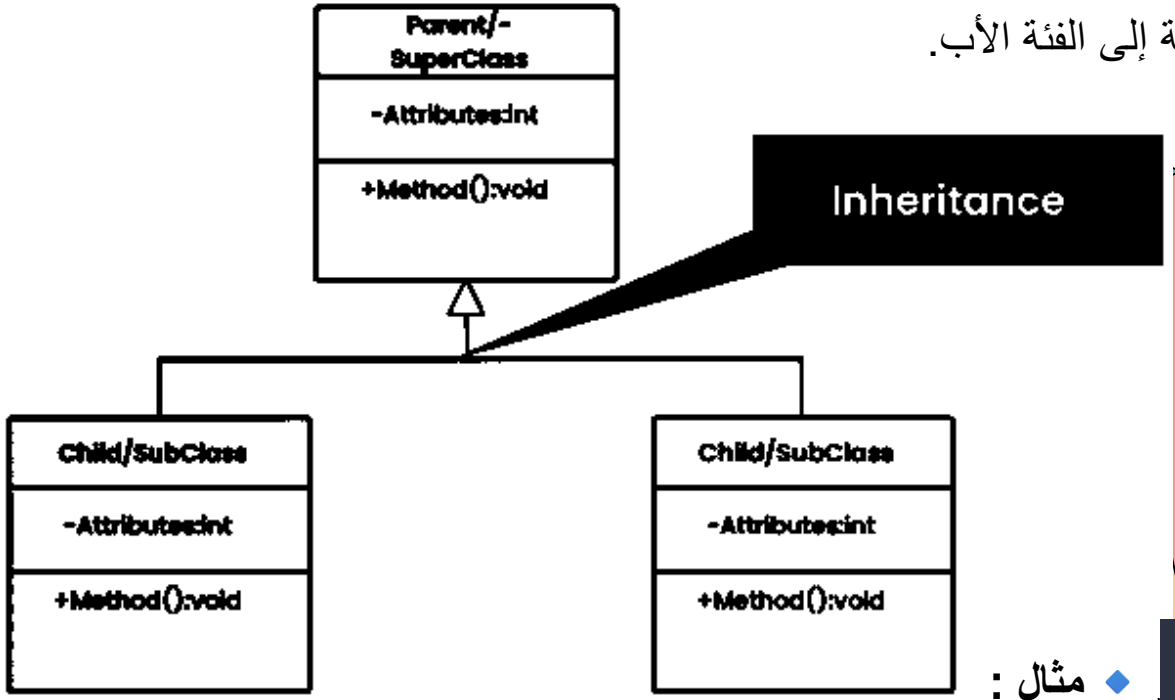


❑ Inheritance:

- Inheritance is also called generalization.
- Hierarchical relationship between classes, where a subclass inherits attributes and methods from its parent class
- Inheritance is represented by a solid line and the inheritance arrowhead.

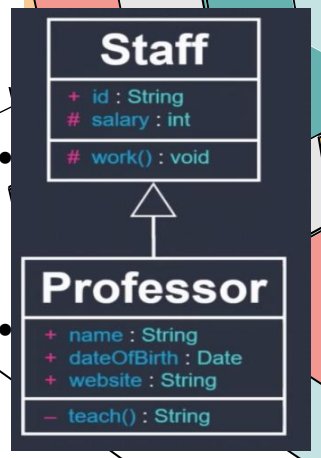
تُعرف الوراثة أيضاً بمفهوم **التعميم** Generalization ، وهي علاقة هرمية بين الفئات Classes في البرمجة الكائنية التوجه. في هذه العلاقة، تقوم الفئة الفرعية Subclass بوراثة السمات Attributes والأساليب Methods من الفئة الأب Parent Class ، مما يتيح إعادة استخدام الشفرة البرمجية وتوسيعها بطريقة منظمة.

يُعبَّر عن الوراثة في مخطط الفئات Class Diagram باستخدام خط متصل ينتهي بسهم مثلثي مجوّف يشير من الفئة الفرعية إلى الفئة الأب.



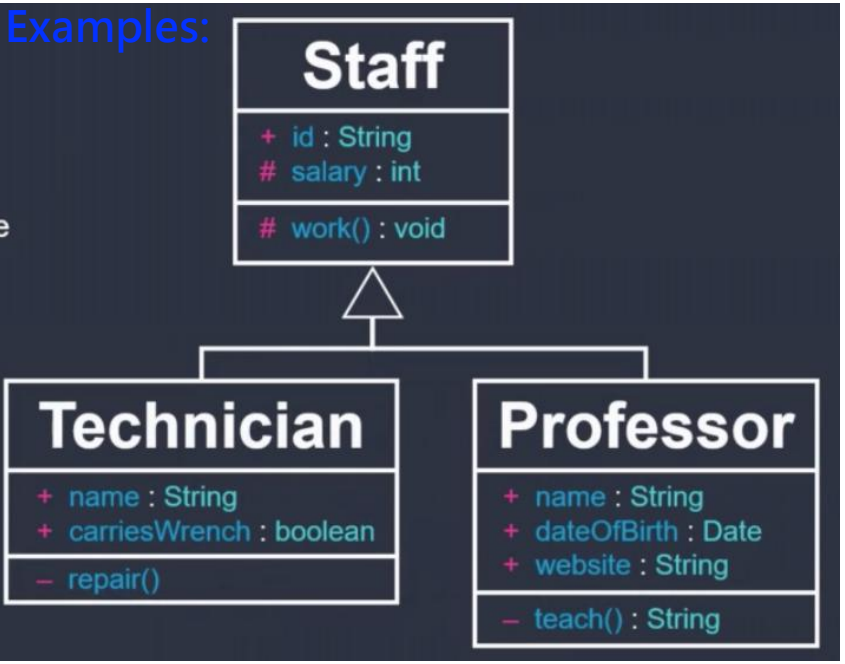
◆ **مثال :**

- فئة Staff **طاقم العمل** تمثل الفئة العامة التي تحتوي على سمات مثل: الاسم، الرقم الوظيفي، والعنوان، بالإضافة إلى أساليب مثل: تسجيل الحضور.
- فئة Professor **بروفسر** تمثل فئة فرعية ترث من الفئة Staff، وتضيف سمات وأساليب خاصة بها مثل: التخصص الأكاديمي، وإعطاء المحاضرات



Relationships between Classes

- Inherit from the same superclass: Multiple classes can inherit from the same superclass.



في البرمجة الكائنية التوجه، يمكن لعدة فئات فرعية Subclasses أن ترث من نفس الفئة الأب Superclass. هذا المفهوم يُعرف بالتعميم Generalization ، حيث تُشارك الفئات الفرعية السمات Attributes والأساليب Methods المعرّفة في الفئة الأب، مع إمكانية إضافة خصائص وسلوكيات خاصة بها.

◆ **مثال أول: فئة Staff**

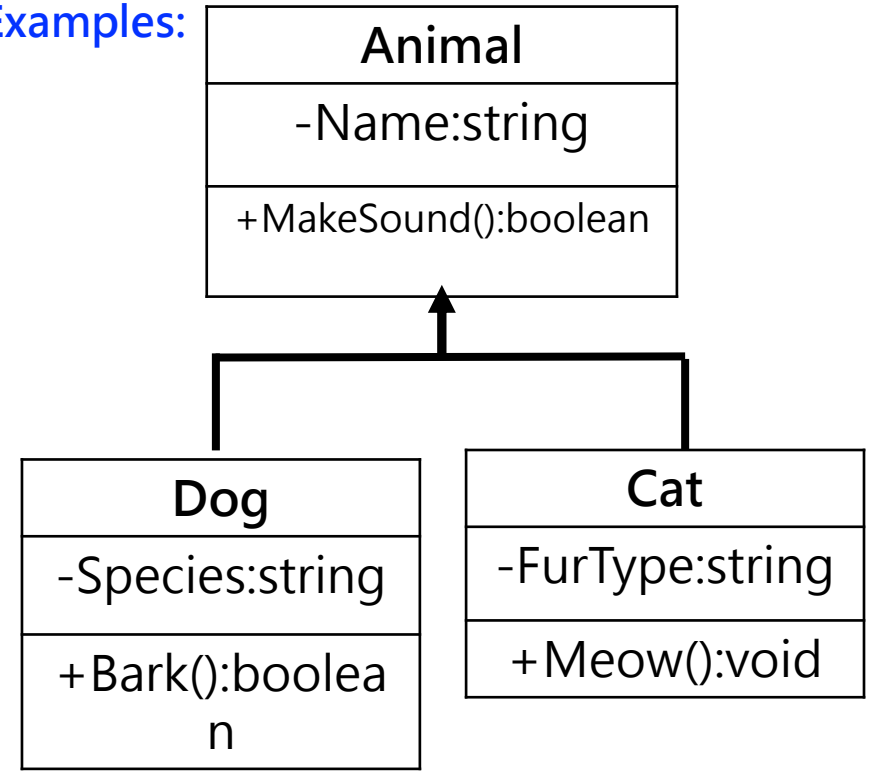
- Staff **طاقم العمل** : الفئة الأب التي تحتوي على سمات عامة مثل الاسم والرقم الوظيفي، وأساليب مثل تسجيل الحضور.
- Professor **بروفسور** : فئة فرعية ترث من Staff وتضيف سمات مثل التخصص الأكاديمي، وأساليب مثل إعطاء المحاضرات.
- Technical: فئة فرعية أخرى ترث من Staff وتضيف سمات مثل المجال الفني، وأساليب مثل صيانة الأجهزة.

Relationships between Classes

- Inherit from the same superclass: Multiple classes can inherit from the same superclass.

في البرمجة الكائنية التوجه، يمكن لعدة فئات فرعية Subclasses أن ترث من نفس الفئة الأب Superclass. هذا المفهوم يُعرف بالتعميم Generalization ، حيث تُشارك الفئات الفرعية السمات Attributes والأساليب Methods المعرّفة في الفئة الأب، مع إمكانية إضافة خصائص وسلوكيات خاصة بها.

Examples:



مثال ثانٍ: فئة Animal

Animal **حيوان :** الفئة الأب التي تحتوي على سمات عامة مثل العمر والوزن، وأساليب مثل الأكل أو النوم.

Dog **كلب :** فئة فرعية ترث من Animalوتضيف سمات مثل السلالة، وأساليب مثل النباح.

Cat **قط :** فئة فرعية أخرى ترث من Animalوتضيف سمات مثل اللون، وأساليب مثل المواء.

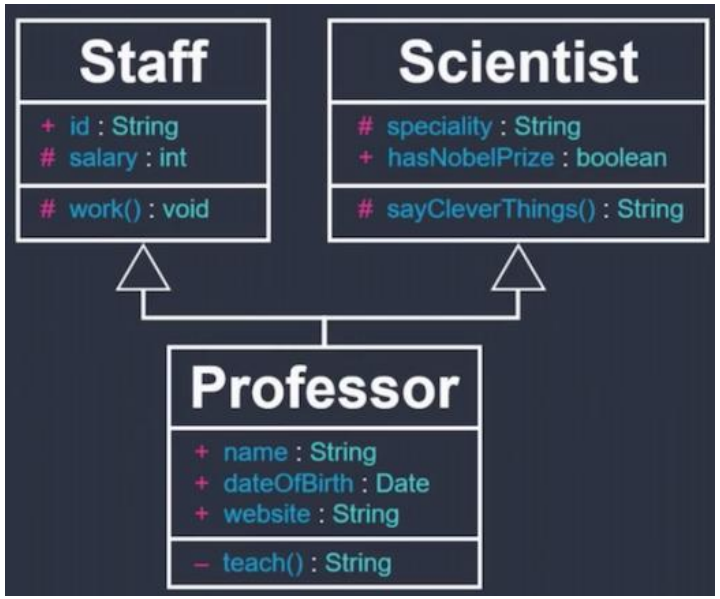
بهذا الشكل، نرى أن الوراثة من نفس الفئة الأب تسمح بإنشاء هيكل هرمي منظم، حيث تُشارك الفئات الفرعية الخصائص العامة، وتُضيف خصائص وسلوكيات متخصصة خاصة بها.

Relationships between Classes
Inheritance Relationship

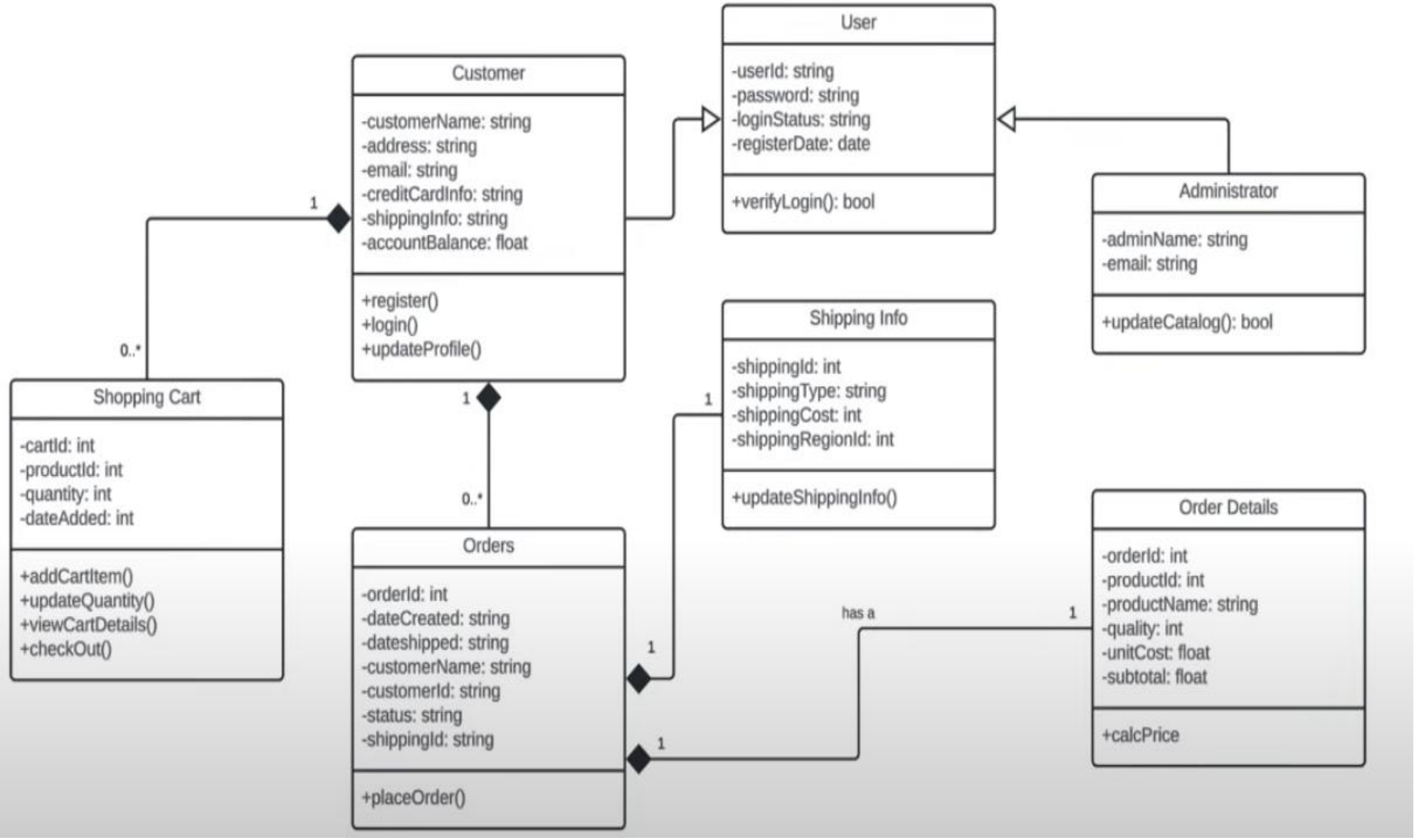
- Multiple inheritance: In UML one subclass can inherit from multiple superclasses. HOWEVER, this is not possible in all programming languages.

في لغة النمذجة الموحدة UML يمكن للفئة الفرعية Subclass أن ترث من أكثر من فئة أب Superclasses في الوقت نفسه. هذا المفهوم يُعرف بالوراثة المتعددة، ويُستخدم لتوضيح أن الفئة الفرعية يمكنها أن تجمع السمات Attributes والأساليب Methods من عدة فئات عليا.

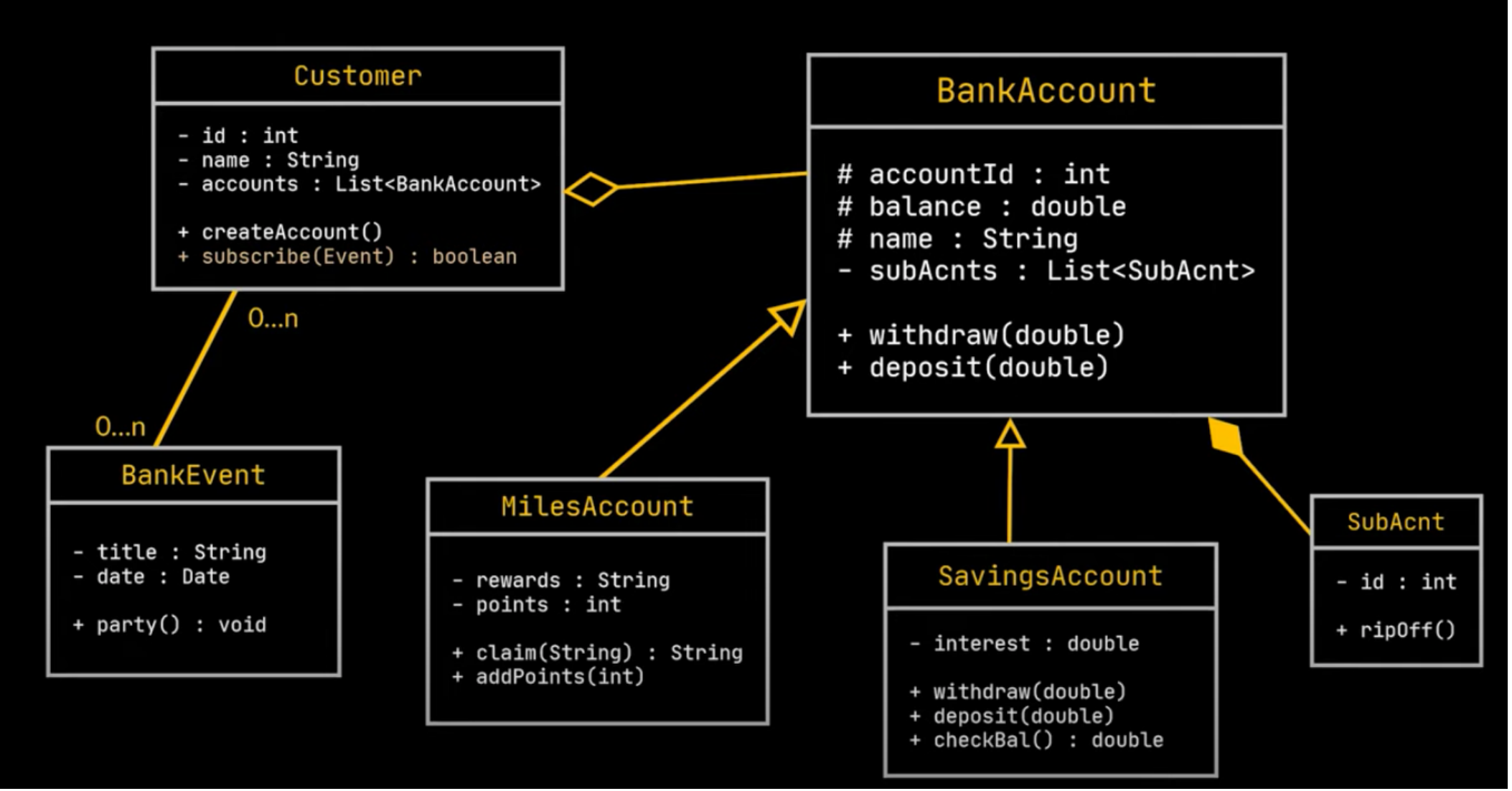
ومع ذلك، يجب التنويه إلى أن هذه الميزة ليست مدعومة في جميع لغات البرمجة؛ فبعض اللغات مثل Java لا تسمح بالوراثة المتعددة بشكل مباشر، بينما لغات أخرى مثل C++ تدعمها.



- **مثال توضيحي:** Staff – Professor – Scientist
  - Staff : فئة عامة تحتوي على سمات مثل الاسم والرقم الوظيفي، وأساليب مثل تسجيل الحضور.
  - Professor: فئة عامة أخرى تحتوي على سمات مثل التخصص الأكاديمي، وأساليب مثل إعطاء المحاضرات.
  - Scientist : فئة عامة ثالثة تحتوي على سمات مثل مجال البحث، وأساليب مثل إجراء التجارب.
  - ResearchProfessor : فئة فرعية ترث من كل من Staff وProfessor وScientist، لتجمع بين خصائص الطاقم الأكاديمي والبحث العلمي.

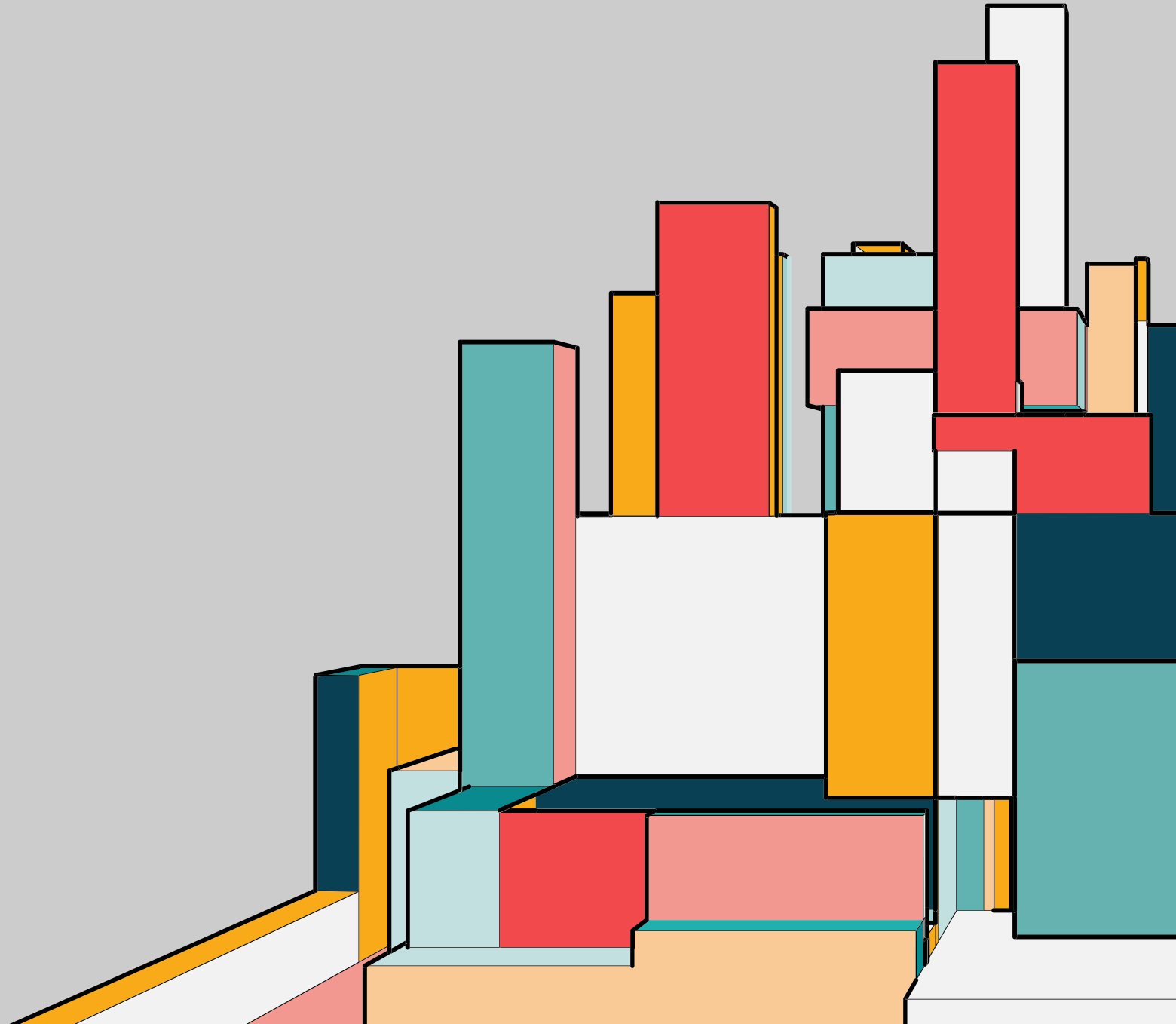

User
-userId: string
-password: string
-loginStatus: string
-registerDate: date
+verifyLogin(): bool
Customer
-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
Administrator
-adminName: string
-email: string
+updateCatalog(): bool
Shipping Info
-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int
+updateShippingInfo()
Shopping Cart
-cartId: int
-productId: int
-quantity: int
-dateAdded: int
+addCartItem()
+updateQuantity()
+viewCartDetails()
+checkOut()
Orders
-orderId: int
-dateCreated: string
-dateshipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string
+placeOrder()
Order Details
-orderId: int
-productId: int
-productName: string
-quality: int
-unitCost: float
-subtotal: float
+calcPrice
has a
1
0..*
1
0..*
1
1
1
1

Example 1:


Customer
- id : int
- name : String
- accounts : List<BankAccount>
+ createAccount()
+ subscribe(Event) : boolean
BankAccount
accountId : int
balance : double
name : String
- subAcnts : List<SubAcnt>
+ withdraw(double)
+ deposit(double)
0...n
0...n
BankEvent
- title : String
- date : Date
+ party() : void
MilesAccount
- rewards : String
- points : int
+ claim(String) : String
+ addPoints(int)
SavingsAccount
- interest : double
+ withdraw(double)
+ deposit(double)
+ checkBal() : double
SubAcnt
- id : int
+ ripOff()

Example 2:

APPENDIXES



ACRONYMS

رقم	الاختصار	الكامل	وصف
1	API	Application Programming Interface	
2	AOD	Aspect-Oriented Design	
3	CBD	Component-Based Design	
4	CRC	Class Responsibility Collaborator (Or Collaboration)	
5	DFD	Data Flow Diagram	
6	DSL	Domain-Specific Language	
7	ERD	Entity Relationship Diagram	
8	FOSS	Free And Open Source Software	
9	IDL	Interface Description Language	
10	MBD	Model-Based Design	
11	MDD	Model-Driven Design	
12	OO	Object-Oriented	
13	PDL	Program Design Language	
14	SDD	Software Design Description	
15	SoC	Separation of Concerns	
16	UML	Unified Modeling Language	

REFERENCES

عنوان المحتوى	المؤلف	سنة النشر	رقم المصدر
Guide to the Software Engineering Body of Knowledge v4.0	Hironori Washizaki	2025	1
Software architecture and design for modern large-scale systems	Marwa Solla	2023	2
الفرق بين المتطلبات الوظيفية وغير الوظيفية وتقنيات استنباطهما وأفضل الممارسات	بكه للتعليم akkah.com	2024	3
Introduction to Software Engineering Design Processes, Principles, and Patterns with UML2	Christopher John Fox	2006	4
ISO/IEC 12207 - Wikipedia	Wikipedia	2025	5

ASSIGNMENT 1

- The previous questions helps in defining the scope of that project. Answers helps in having a clear consensus about if this production project could be conducted or not.
- If there some technical, legal, expertise- related, political, or other could arise, and couldn't be mitigated, then the production project is not feasible.
- If the dedicated budget was not reasonable at all then, Boeing will notify Libyan Airline that this plane couldn't be produced and delivered to Libya.
- In contrary, if the project shows feasibility then, the rest of information will help in defining the scope of the project.