

Unified Modeling Language

Plan:

1. Overview
2. Types of Diagrams
3. Zoom on Classes Diagrams

Overview

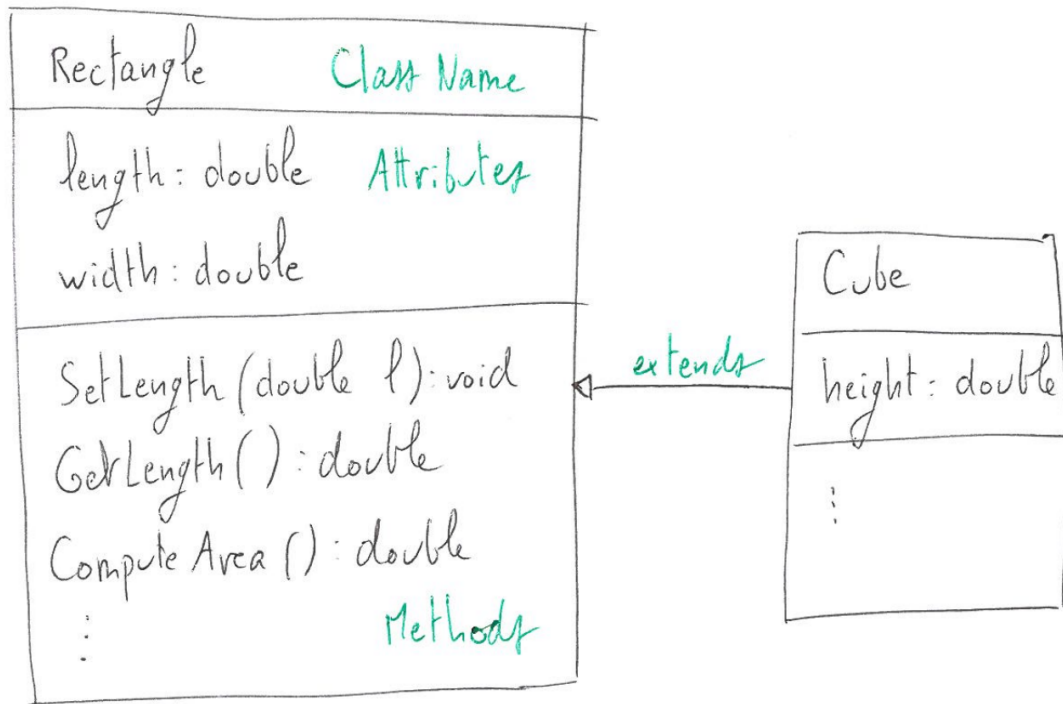
One approach for analysis, design, implementation and deployment of databases and their applications. Databases interact with multiple softwares and users, we need a common language.

Unified Modeling Language (<http://uml.org>) is a *standard*:

- Generic
- Language-independent
- Platform-independent

Wide, powerful, but also intimidating.

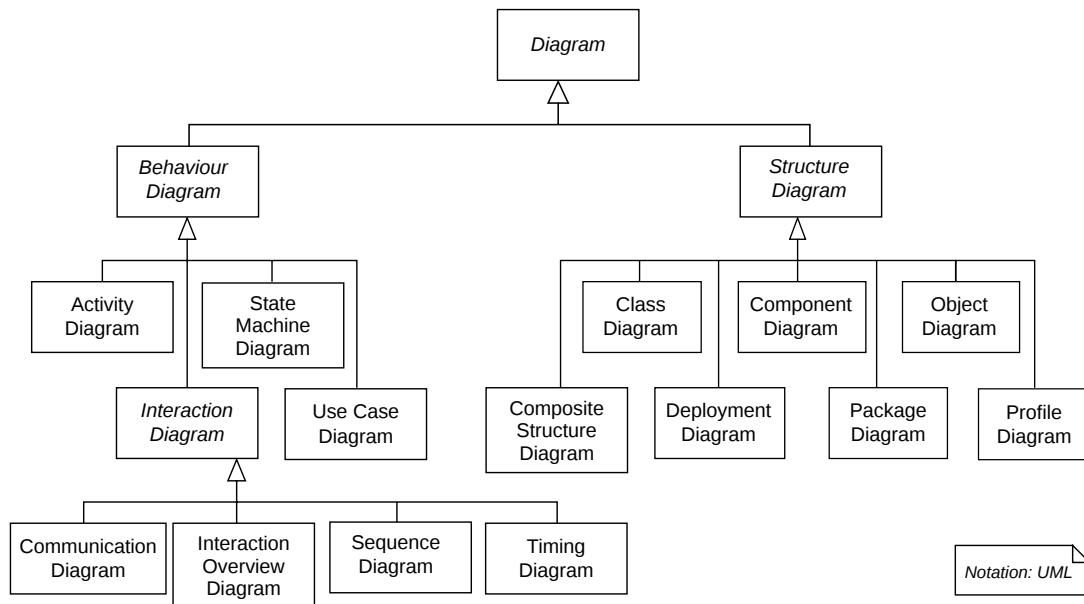
You know UML from object-oriented programming language:



That's a class diagram, there are other types of diagrams, they are not unrelated! For instance, using communication diagrams, deployment diagrams, and state chart diagrams, you can collect the requirements needed to draw a class diagram! They each offer a viewpoint on a software that will help you in making sure the various pieces will fit together: it is a tool commonly used in software engineering, and useful in database design.

Types of Diagrams

There are 14 different types of diagrams, divided between two categories: structural and behavioral.



Structural UML diagrams

They describe structural, or static, relationships between objects, softwares.

- **Class diagram** describes static structures: classes, interfaces, collaborations, dependencies, generalizations, etc. We can represent conceptual data base schema with them!
- **Object diagram**, a.k.a. instance diagram, represents the static view of a system at a particular time. You can think of a “freeze” of a program, to be able to observe the value of the variables and the objects (or instances) created.
- **Component diagram** describes the organization and the dependencies among software components (e.g., executables, files, libraries, etc.), to describe how an arbitrary large software system is split into pieces.

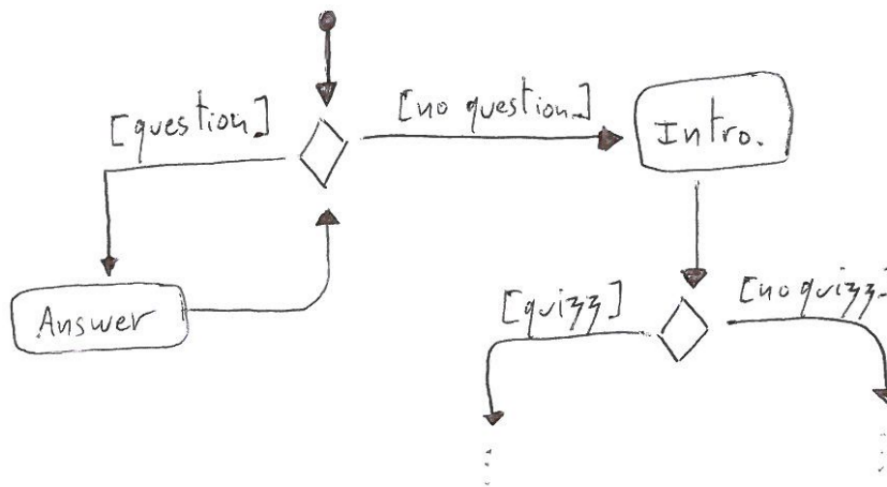
- **Deployment diagram** is the description of the physical deployment of artifacts (i.e., software components) on nodes (i.e., hardware). If your program runs on a local computer, fetching data from the Internet, and storing output on a server, you may describe this situation using this sort of diagram.

In this category also exist **Composite structure diagram**, **Package diagram** and **Profile diagram**.

Behavioral UML diagrams

They describe the behavioral, or dynamic, relationship, between components.

- **Use case diagram** describes the interaction between the user and the system. Supposedly, it is the privileged tool to communicate with end-users.
- **State machine diagram**, a.k.a., state chart diagram, describes how a system react to external events. You can picture yourself a complex form of finite state automata diagram.
- **Activity diagram** is a flow of control between activities. You may have seen them already, they are supposedly easy to follow:



Then there is the sub-category of “Interaction diagrams”:

- **Sequence diagram** describes the interactions between objects over time, the flow of information or messages between objects. It is helpful to grasp the time ordering of the interactions.
- **Communication diagram**, a.k.a., collaboration diagram, describes the interactions between objects as a serie of sequenced messages. It is helpful to grasp the structure of the objects, who is interacting with who.

This sub-category also comprise **Timing diagram** and **Interaction overview diagram**.

Zoom on Classes Diagrams

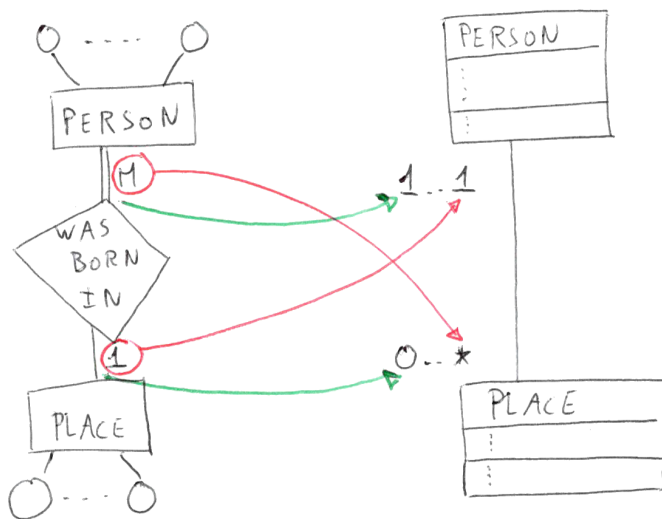
Looking at the “COMPANY conceptual schema in UML class diagram notation”, and comparing it with the “ER schema diagram for the COMPANY database” from the textbook, can help you in writing your own “Rosetta Stone” between ER and UML diagram. Let us introduce some UML terminology for the class diagrams.

UML	ER
Class	Entity Type
Class Name	Entity Name
Attributes	Attributes
Operations (or Method)	<i>Sometimes</i> Derived Attributes
Association	Relationship Type
Link	Relationship Instance
Multiplicities	Structural Constraint

As well as for ER diagram, the domain (or data type) of the attributes is optional. A composite attribute in a ER diagram can be interpreted as a structured domain in a UML diagram (think of a `struct`), and a multi-valued attribute requires to create a new class.

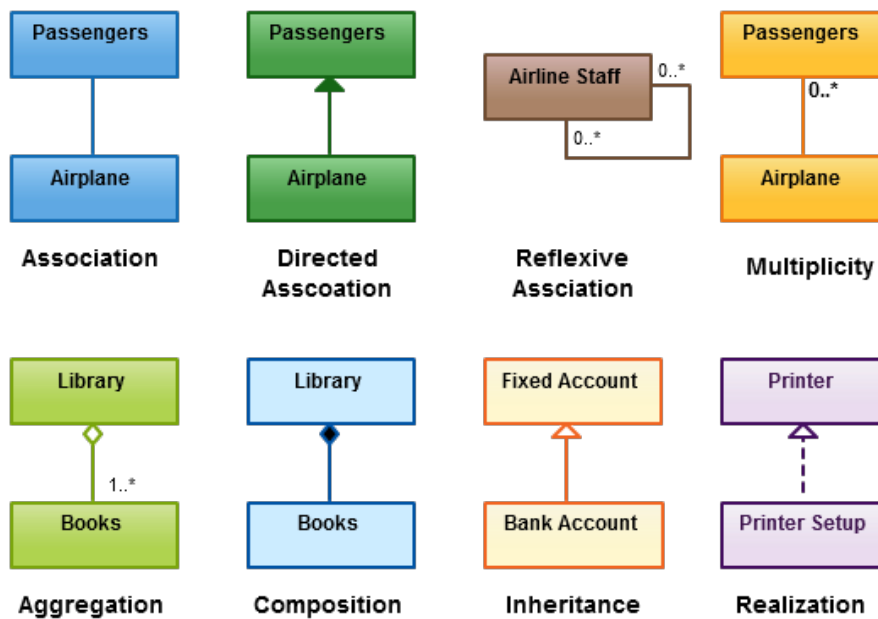
Associations are, to some extend, more expressive than relationship types:

- **As for relationship types**, they can be recursive (or reflexive), and uses role names to clarify the roles of both parties.
- **As for relationship types** they can have attributes: actually, a whole class can be connected to an association.
- **As for relationship types**, they can express a cardinality constraint on the relation between classes. They are written as `min .. max`, with `*` for “no maximum”, and the following shorthands: `*` stands for `0 .. *` and `1` stands for `1 .. 1`. An association with `1` on one side and `*` on the other (resp. `1` and `1`, `*` and `1`, `*` and `*`) is sometimes called “one-to-many” (resp., “one-to-one”, “many-to-one”, “many-to-many”). The notation is partially inverted w.r.t. ER diagrams:



- As opposed to the relationship types, they can have a direction, indicating that the user should be able to navigate them only in one direction, or in two (which is the default). This is used for security or privacy purposes.
- As opposed to the relationship types, they come in various flavors:
 - You can express *aggregation*, a.k.a. “is part of” relationship, between a whole object and its component (that have their own existence).
 - You can express *composition*, which is the particular case of aggregation where the component doesn’t have an existence of their own.
 - You can express *generalization*, a.k.a. inheritance, that eliminates redundancy and makes a class a *specialization* of another one.
- As opposed to the relationship types, they can be *qualified*, implying that a class is not connected to the other class as a whole, but to one particular attribute, called *the qualifier*, or *discriminator*.

This last feature can be used for weak entities, but not only.



Some of those subtleties depend on your need, and are subjective, but are important tool to design properly a database, and relieving the programmer from the burden of figuring out many details.

Sources:

- https://en.wikipedia.org/wiki/Unified_Modeling_Language
- <https://creately.com/blog/diagrams/class-diagram-relationships/>
- Section 3.8 (7th Edition) or 7.8 (6th Edition) of your textbook.