KAROLY NYISZTOR

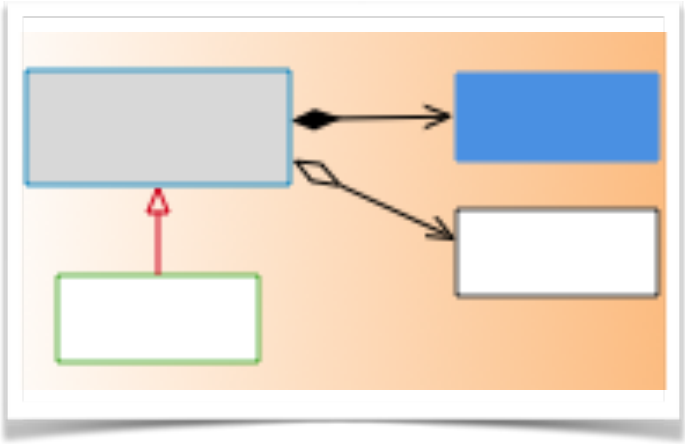
WWW.LEAKKA.COM

CN-2023-0008

Class Diagram Cheat Sheet

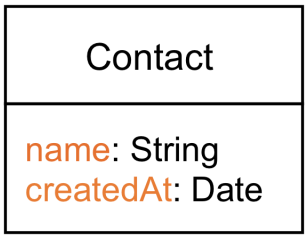
A practical overview of the Class UML diagram

The Class Diagram



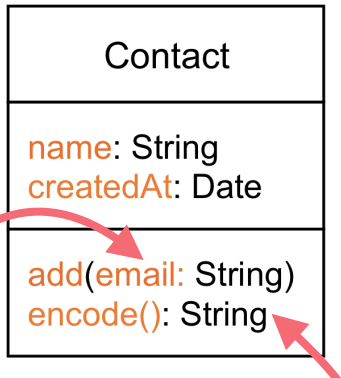
Purpose: visualize the structure of a software system

To **represent a class**, we draw a rectangle with the name of the class in it.



List the class attributes in a separate compartment. The attribute's name and data type are separated by a colon.

The operations compartment holds **the methods of the class**.



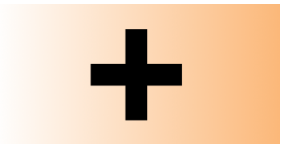
Method parameters appear within the parenthesis as name-data type pairs.
If a method has a **return value**, add a colon after the closing parenthesis followed by the return type.

CLASS ATTRIBUTES AND METHODS

Class names should be nouns in UpperCamelCase (e.g., "FileManager," "PersistentStore").
The class attributes should be nouns, and method names should be verbs. Use the lowerCamelCase format for attribute and method names. Examples: "creationDate", "update()" etc.

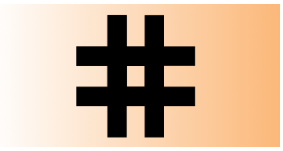
Visibility Levels

Purpose: control who can access the attributes and the methods of our class

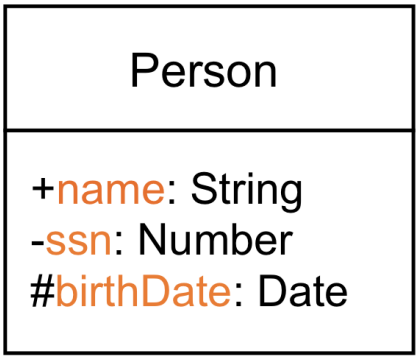


Public
A class method or attribute marked as public can be used by code outside of the object

Private
Private attributes and methods can only be used within the class that defines them



Protected
Only child classes and the defining class can access protected attributes or methods.



Example

The Person class has three attributes:

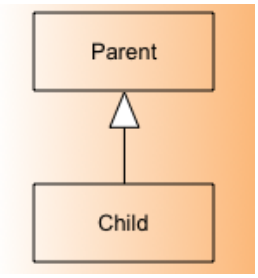
- *name* is public
  Any code inside and outside the Person class can access it.
- *ssn* is private
  Only methods in the Person class can access this attribute.
- *birthDate* has protected visibility
  That means that it's only visible to the Person class and its child classes.

PUBLIC, PROTECTED OR PRIVATE?

While there are no hard rules for choosing the visibility levels, you should avoid making all the attributes and methods of a class public; expose only as much as needed and hide everything else.
Class attributes should almost never be public. Instead, provide getters and setters to access your class's data.

Relationships

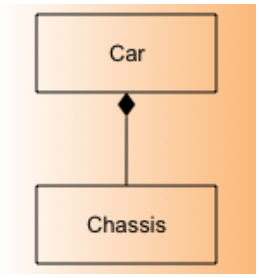
Purpose: represent the relations and the dependencies between the objects



Generalization

“is-a”

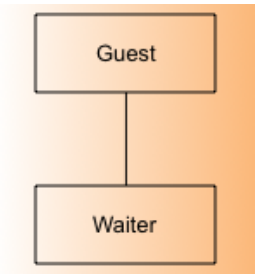
Represents that one element is based on another element.



Composition

“part-of”

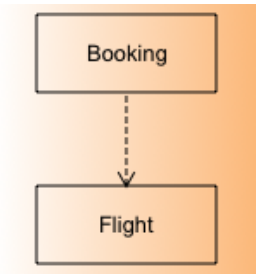
Denotes ownership. When the owning object is destroyed, the contained objects get destroyed, too.



Association

“interaction”

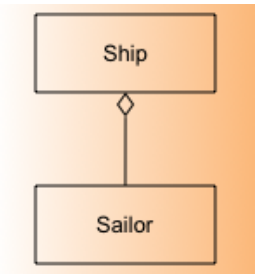
Used to visualize that the elements interact with/refer to each other.



Dependency

“references”

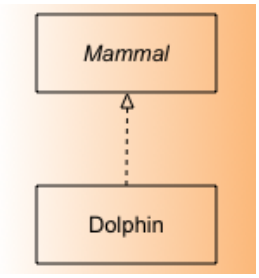
Indicates that one element receives a reference to another element (e.g. via a method parameter).



Aggregation

“has-a”

Represents a part-whole relationship. Parts don’t die with the whole.



Realization

“implements behavior”

Shows that a class implements the behavior specified by another model element.

ASSOCIATION, AGGREGATION OR COMPOSITION?

Associations are useful in the early stages of a design when you want to visualize the relationships between classes quickly.

As your design evolves, you may want to be more specific about these relationships, and start showing directed associations, aggregations, compositions or dependencies.