UML Class Notation

A class represent a concept which encapsulates state (**attributes**) and behavior (**operations**). Each attribute has a type.

* It shows the attributes, classes, functions, and relationships to give an overview of the software system

**components of a Class Diagram**

* **Upper Section-**ClassName
* **Middle Section-**Attributes
* **Lower Section-**Methods

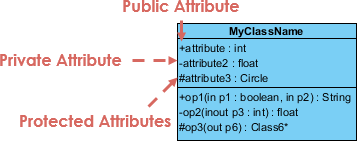
**A class notation represent three parts and we enclose them in a rectangle.**

* **Upper Section(**ClassName ) Every class must have a name and the name of the class appears in the top and first part of the rectangle.
* **Middle Section(**Attributes)- The second part of the class is its attributes which is used to represent the property of the class. It is written in the middle portion od rectangle and the data type of attributes is written after colon.
* **Lower Section(**Methods)-Operation are shown in the third partition. It shows the actual function performed on the class. The + operator is written before class operation.

|  |
| --- |
| **Library** |
| -Library reg\_no :int  -Library name: string  -Address: string |
| +Add book  +Remove book |

### **Class Visibility**

The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

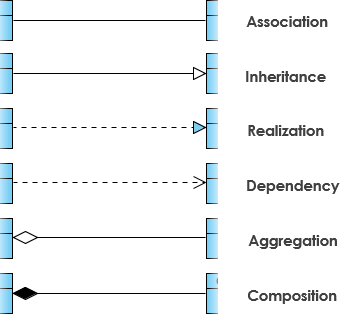


* + denotes public attributes or operations
* - denotes private attributes or operations
* # denotes protected attributes or operations

NORMALLY SABAI ATTRIBUTES PRIVATE FUNCTIONS/METHODS PUBLIC HUNE RAEXA

## Relationships between classes

. A class may be involved in one or more relationships with other classes. A relationship can be one of the following types:



Dependency vneko tyo class ko properties le chai as an parameter arko class ko object use gariraxa

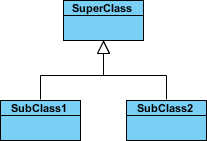
Composition live and die with that class

Realization rw inheritance ustae ustae ho so symbol ni tei inheritance wala lai --- ani color

1. Inheritance

* Represents an "is-a" relationship.

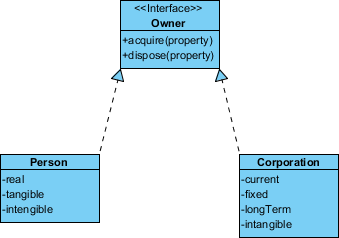
SubClass1 and SubClass2 are derived from SuperClass.



**Realization**

Realization is a relationship between the blueprint class and the object containing its respective implementation level details.

For example, the Owner interface might specify methods for acquiring property and disposing of property. The Person and Corporation classes need to implement these methods, possibly in very different ways.



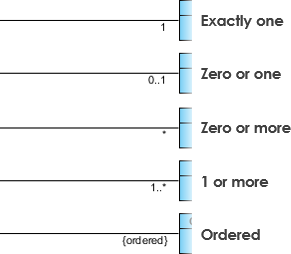
### **2. Simple Association**

* A structural link between two peer classes.

### **Cardinality**

Cardinality is expressed in terms of:

* one to one
* one to many
* many to many



### 3. **Aggregation**

A special type of association.

* It represents a "part of" relationship.
* Class2 is part of Class1.

Aggregation

### **Composition**

* A special type of aggregation where parts are destroyed when the whole is destroyed.
* Objects of Class2 live and die with Class1.

Composition

### **Dependency**

An object of one class might use an object of another class in the code of a method. If the object is not stored in any field, then this is modeled as a dependency relationship.

* A special type of association.
* Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).

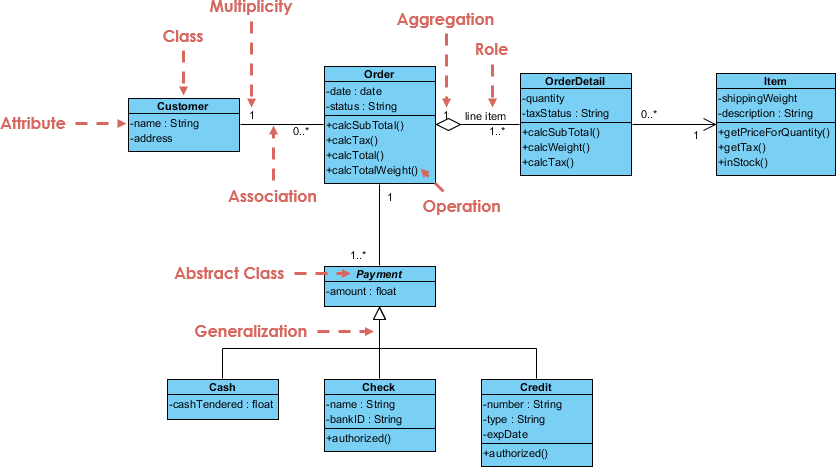
Example:

* The Person class might have a hasRead method with a Book parameter that returns true if the person has read the book (perhaps by checking some database).

Dependency

## class Diagram Example: Order System (copy ma herne rough wala )

## order detail chai part of order vyoni tw so aggregation use vairaxa ani check cash credit sabai payment kae different methods haru check is payment esto vyo so generatlization use vairaxa



Object Diagram

* Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.
* Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

Object diagrams are used to render a set of objects and their relationships as an instance

Notation of an object diagram

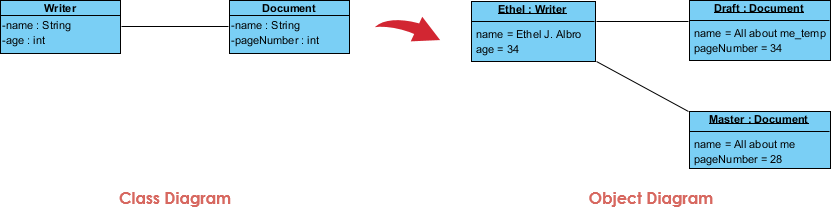
|  |
| --- |
| **Object name: Class** |
| + object attributes |

Example:

|  |
| --- |
| **Apple: Fruit** |
| +color: red  +quantity: 1kg |

|  |  |
| --- | --- |
| **Object Names**:   * Every object is actually symbolized like a rectangle, that offers the name from the object and its class underlined as well as divided with a colon. | Object Diagram Notation: Object |
| **Object Attributes**:Similar to classes, you are able to list object attributes inside a separate compartment. However, unlike classes, object attributes should have values assigned for them. | Object Diagram Notation: Object Attribute |
| **Links:**   * **Links tend to be instances associated with associations. You can draw a link while using the lines utilized in class diagrams.** | Object Diagram Notation: Links |

### **Object Diagram Example III - Writer**



DEPLOYMENT DIAGRAM rw component diagram COPY KO HERNE ARU PARDENW