REPUBLIC OF CAMEROON
*Peace- Work- Fatherland*

UNIVERSITY OF BAMENDA

NATIONAL HIGHER POLYTECHNIQUE INSTITUTE (N.A.H.P.I.)
BAMBILI – BAMENDA
**********
Box 39 Bambili, CAMEROON

Tel: +237 74 94 65 24 Fax:+237 33 05 10 69

REPUBLIQUE DU CAMEROUN
*Paix- Travail- Patrie*

UNIVERSITE DE BAMENDA

ECOLE NATIONALE SUPERIEURE POLYTECHNIQUE (E.N.S.P.)
BAMBILI – BAMENDA
**********
Box 39 Bambili, CAMEROUN

Tel: +237 74 94 65 24 Fax:+237 33 05 10 69

# OBJECT ORIENTED DESIGN UML BASIC NOTATION

**COURSE TITLE**: OBJECT ORIENTED DESIGN AND ANALYSIS

**COURSE CODE**: **COME3101**

# Group Members

| Name | Matricule | Participation |
|---|---|---|
| LOWEH KENOLY FONYUY | UBA21E0029 | |
| MBA VERON MBA | UBA21E0030 | |
| MBAH EMMANUEL ANCHINBOM | UBA21E0031 | |
| MBAH LOIS LA GRACE | UBA21E0032 | |
| MC-DILLAN FRU NGWA | UBA21E0033 | |
| MENDOUGA NGANDI MAJOIE DESIRE | UBA21E0034 | |
| MENYAM NCHIFOR | UBA21E0035 | |
| MOFIRO MOUGANG JEAN CHUNGONG | UBA21E0036 | |
| MOFOR-REBE GODLOVE CHE | UBA21E0037 | |
| MUNKI BRANDOLEE AMANWI | UBA21E0427 | |

# Table of Content

# OOAD – UML Basic Notations

The Unified Modeling Language (UML) is a graphical language for OOAD that gives a standard way to write a software system's blueprint. It helps to visualize, specify, construct, and document the artifacts of an object-oriented system. UML defines specific notations for each of its building blocks. These building blocks are given as follows;

## 1. Class

One of the most important elements in UML is the class, which is a fundamental building block in object-oriented programming, representing a collection of objects with the same attributes, behavior, and relationships. In this article, we will explain the basic notation of classes in UML, including their structure, notation, and usage.

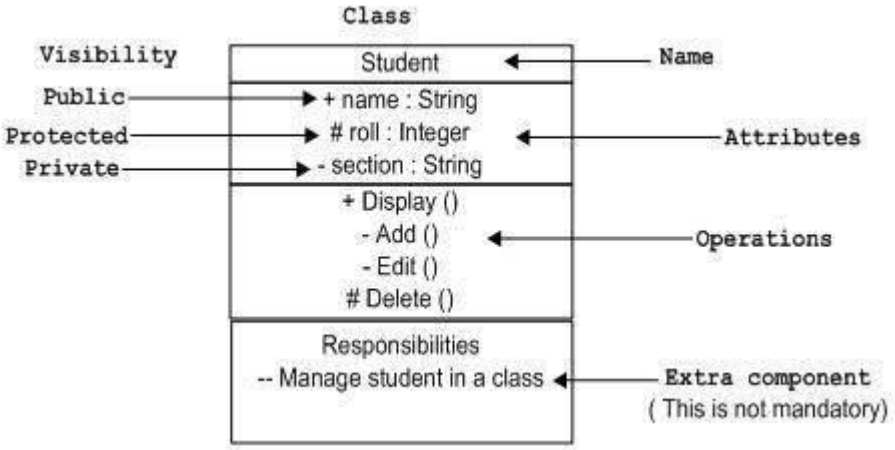
The structure of a class in UML consists of three compartments:

- the top compartment contains the name of the class and it is written in bold.
- the middle compartment lists the attributes of the class
- the bottom compartment lists the methods or operations of the class.

The visibility of the attributes and operations(methods) can be represented in the following ways:

- **Public**: A public member is visible from anywhere in the system. In class diagram, it is prefixed by the symbol '+'.
- **Private**: A private member is visible only from within the class. It cannot be accessed from outside the class. A private member is prefixed by the symbol '–'.
- **Protected**: A protected member is visible from within the class and from the subclasses.

The notation of a class in UML is a rectangle with three compartments, as shown in the following example:




The usage of classes in UML is to represent the structure and behavior of the system being modeled. Classes can be used to represent the entities of the system, their attributes and operations, and the relationships between them. Classes can also be used to represent the interfaces of the system, the components that implement them, and the packages that contain them. Classes can be connected to other classes and objects in a UML diagram using various types of relationships, such as association, aggregation, composition, inheritance, and realization.

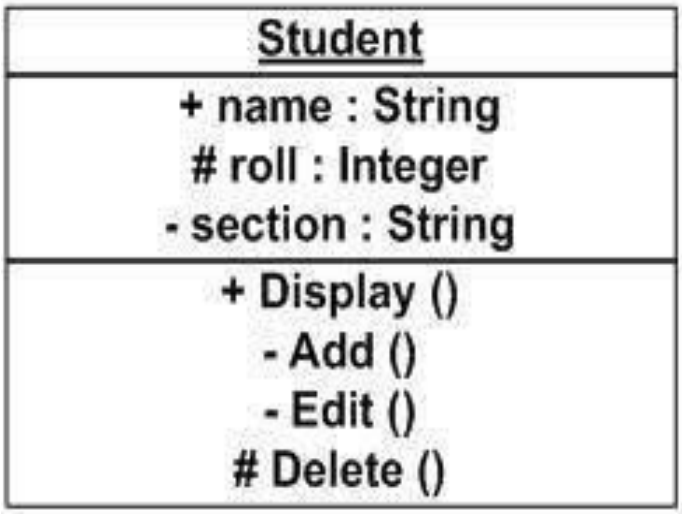
Classes are a powerful and flexible tool for modeling software systems in UML, as they allow us to represent the structure and behavior of the system in a clear and concise way. By using classes, we can model the entities and relationships of the system, and document the design of the system in a standardized and easily understood format.

## 2. Object

One of the most important elements in UML is the object, which is an instance of a class, representing a specific entity with its own unique characteristics and behavior.

The structure of an object in UML consists of a single compartment, containing the name of the object and the name of the class it belongs to. The name of the object is written in bold and is placed at the top of the rectangle, followed by a colon and the name of the class. The name of the class is written in italics and is placed below the name of the object.

Using the example above, the notation of an object in UML is as shown in below;




The usage of objects in UML is to represent the specific instances of a class in a system, and to show how they relate to each other and to the class they belong to. Objects can be used to represent the real-world entities that a system is designed to model, such as people, places, and things. Objects can also be used to represent the abstract concepts that a system is designed to implement, such as algorithms, processes, and data structures.

Objects are a useful and important element of UML, as they allow us to represent the specific instances of a class in a system, and to show how they relate to each other and to the class they belong to. By using objects, we can model the specific entities and relationships of a system, and document the design of the system in a standardized and easily understood format.

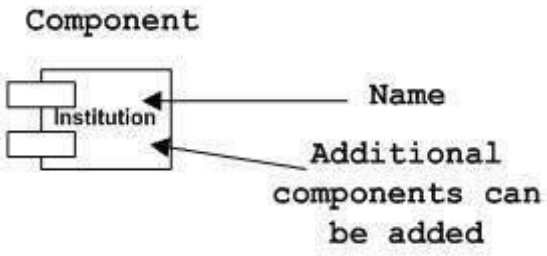
## 3. Component

A component is a modular, self-contained unit of an application, representing a piece of software that can be independently developed, tested, and deployed. It is a physical and replaceable part of the system that conforms to and provides the realization of a set of interfaces

The structure of a component in UML consists of two compartments: the left compartment contains the name of the component, and the right compartment

lists its interfaces. The name of the component is written in bold and is placed at the top of the rectangle, followed by a colon and the visibility of the component (public, private, or protected). The interfaces of the component are listed in the right compartment, each one written on a separate line and preceded by its name and visibility.

The notation of a component is as shown below:




The usage of components in UML is to represent the modular structure of a system, and to show how the different parts of the system interact with each other. Components can be used to represent the physical or logical units of an application, such as libraries, executables, or services. Components can also be used to represent the interfaces that a system exposes to its users or to other systems, and the dependencies between them.

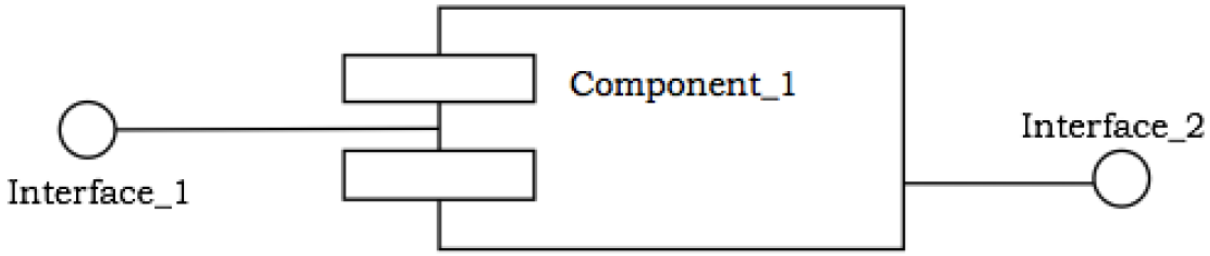
Components are a useful and important element of UML, as they allow us to represent the modular structure of a system, and to show how the different parts of the system interact with each other. By using components, we can model the structure and behavior of a system, and document the design of the system in a standardized and easily understood format.

## 4. Interface

One of the basic concepts in UML is the interface. An Interface is a collection of methods of a class or component. It specifies the set of services that may be provided by the class or component.

Generally, an interface is drawn as a circle together with its name. An interface is almost always attached to the class or component that realizes it.

The following figure gives the notation of an interface.:




The lollipop represents the operations or behaviors that are defined by the interface, and the circle symbol represents the interface itself. The operations are typically shown as small squares, with the name of the operation written inside.

There are several types of interfaces in UML, including required interfaces and provided interfaces.

- **A required interface** is an interface that a class or component expects another class or component to implement.
- **A provided interface** is an interface that a class or component offers to other classes or components.

**For example**, consider a simple system that consists of a class called "Person" and a class called "Bank". The Person class might have a required interface called "Withdraw Money" that specifies an operation called "withdraw()" that takes an amount as an input and returns a boolean value indicating whether the withdrawal was successful. The Bank class might have a provided interface called "Deposit Money" that specifies an operation called "deposit()" that takes an amount as an input and returns a boolean value indicating whether the deposit was successful.

In this example, the Person class would depend on the Bank class to implement the "Deposit Money" interface, and the Bank class would depend on the Person class to implement the "Withdraw Money" interface. This relationship is known as a dependency, and is represented in UML by a dashed line with an arrow pointing from the dependent class to the class that it depends on.

There are many other aspects of UML notation that are related to interfaces, such as the use of stereotypes, constraints, and multiplicity.

- **Stereotypes** are used to specify the type of interface.
- **constraints** are used to specify any additional rules or conditions that apply to the interface.

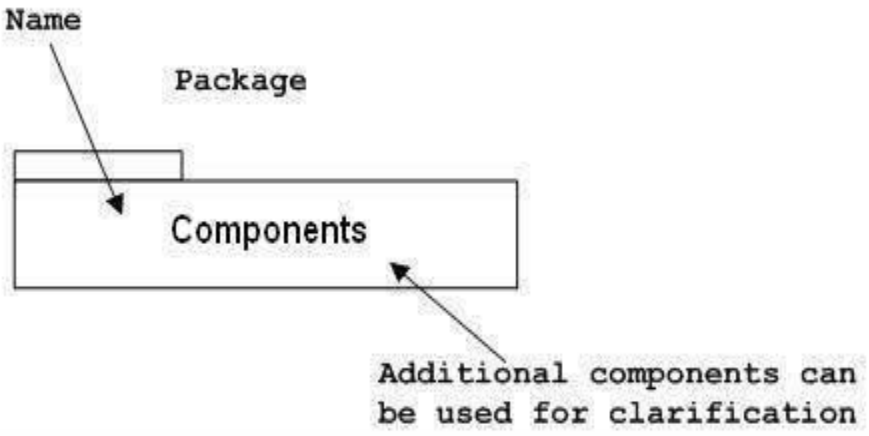
- **Multiplicity** is used to specify the number of instances of a class or component that can be connected to the interface.

Overall, the concept of interfaces is an important part of UML notation, and is used to define the relationships and dependencies between different classes and components in a software system.

## 5. Package

In UML, a package is a grouping of related elements, such as classes, interfaces, and other packages. It is used to organize and structure the elements of a software system in a logical and hierarchical manner.

In UML notation, a package is represented by a tabbed folder symbol, as shown in the following diagram:




Inside the package, the various elements that it contains are shown as small rectangles with their names written inside. The package may also contain sub-packages, which are represented by smaller tabbed folder symbols within the main package symbol.

There are several reasons why packages are used in UML. One reason is to provide a logical grouping of related elements. For example, if a software system has a large number of classes, it can be helpful to group them into packages based on their function or purpose. This can make the system easier to understand and navigate, especially for larger and more complex systems.

Another reason for using packages is to help manage the visibility and accessibility of elements within the system. By default, elements within a package are only visible and accessible to other elements within the same package. However, this visibility and accessibility can be modified using various UML notation elements, such as imports, exports, and public, private, and protected keywords.

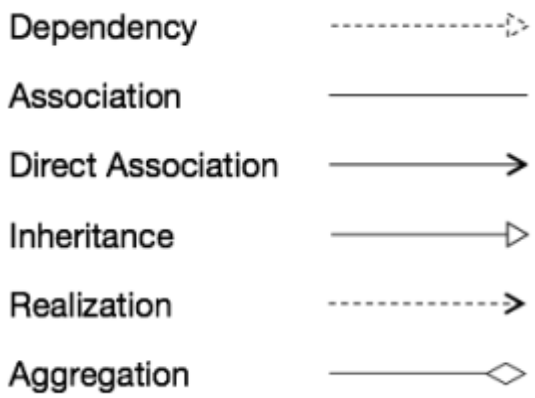
For example, consider a software system that consists of a package called "Accounts" that contains several classes related to financial transactions. The Accounts package might have a sub-package called "Checking" that contains classes related to checking accounts, and another sub-package called "Savings" that contains classes related to savings accounts. Within the Accounts package, the classes in the Checking and Savings sub-packages might be set to private visibility, meaning that they are only visible and accessible within the Accounts package.

In addition to organizing and structuring elements, packages can also be used to represent relationships between elements. For example, a package might contain a class that uses or depends on another class in the same package, or a package might contain a class that implements an interface defined in another package. These relationships can be represented using UML notation elements such as dependencies, associations, and generalizations.

Overall, the concept of packages is an important part of UML notation, and is used to organize, structure, and represent relationships between the various elements of a software system.

## 6. Relationships

In UML, a relationship is a connection or link between two or more elements, such as classes, interfaces, or packages. There are several types of relationships in UML, including dependencies, associations, and generalizations.




- **Dependencies** are the most basic type of relationship in UML. They represent a relationship in which one element (the dependent element) relies on or uses another element (the provider element) in some way. Dependencies are represented in UML notation by a dashed line with an arrow pointing from the dependent element to the provider element.

For example, consider a software system that consists of a class called "Person" and a class called "Bank". The Person class might have a method called "withdraw()" that relies on the Bank class to perform a financial transaction. In this case, the Person class would have a dependency on the Bank class, and this dependency would be represented in UML by a dashed line with an arrow pointing from the Person class to the Bank class.

- **Associations** are a more complex type of relationship in UML. They represent a structural relationship between two classes, in which one class (the source class) has a reference to or uses another class (the target class). Associations are represented in UML notation by a solid line with an arrow pointing from the source class to the target class.

For example, consider a software system that consists of a class called "Customer" and a class called "Order". The Customer class might have an association with the

Order class, in which each instance of the Customer class has a reference to one or more instances of the Order class. In this case, the association would be represented in UML by a solid line with an arrow pointing from the Customer class to the Order class.

- **Generalizations** are another type of relationship in UML. They represent an inheritance or specialization relationship between two classes, in which one class (the subclass) is a specialized version of another class (the superclass). Generalizations are represented in UML notation by a solid line with a hollow triangle arrowhead pointing from the subclass to the superclass.

For example, consider a software system that consists of a class called "Animal" and a class called "Dog". The Dog class might be a specialized version of the Animal class, in which it inherits certain attributes and behaviors from the Animal class but also has some additional characteristics that are specific to dogs. In this case, the Dog class would have a generalization relationship with the Animal class, and this relationship would be represented in UML by a solid line with a hollow triangle arrowhead pointing from the Dog class to the Animal class.

Overall, relationships are an important part of UML notation, and are used to represent the connections and dependencies between different elements in a software system.

## References


- "UML Basics: The Class Diagram" (https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/)
- "*UML - Basic Notations*, n.d."
- "Unified Modeling Language (UML) - Tutorialspoint" (https://www.tutorialspoint.com/uml/index.htm)
- "IBM Documentation"
- "UML 2 Tutorial - Object Diagram" (https://www.uml-diagrams.org/object-diagrams.html)
- "UML Component Diagrams: An Agile Introduction" (https://www.agilemodeling.com/artifacts/componentDiagram.htm)