**Practical 2**

|  |  |
| --- | --- |
| Roll No. A016 | Name: Varun Khadayate |
| Program: B-Tech (CSBS) | Division: SY |
| Batch: 1 | Date of Experiment: 30-01-2021 |
| Date of Submission: 30-01-2021 | Grade: |

**AIM:** To understand various Class Diagram Relationship.

**Theory:**

What are the Class Diagrams?

Class diagrams are the main building block in object-oriented modeling. They are used to show the different objects in a system, their attributes, their operations, and the relationships among them.

|  |
| --- |
| Window |
| size: Size  visibility: boolean |
| display()  hide() |

In the example, a class called “Window” is depicted. Classes in class diagrams are represented by boxes that are partitioned into three:

1. The top partition contains the name of the class.
2. The middle part contains the class’s attributes.
3. The bottom partition shows the possible operations that are associated with the class.

The example shows how a class can encapsulate all the relevant data of a particular object in a very systematic and clear way. A class diagram is a collection of classes like the one above.

### ****Relationships in Class Diagrams****

### Association



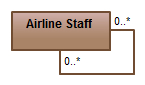
is a broad term that encompasses just about any logical connection or relationship between classes. For example, passenger and airline may be linked as above

### Directed Association



refers to a directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.

Reflexive Association



This occurs when a class may have multiple functions or responsibilities. For example, a staff member working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member. If the maintenance crew member is managed by the aviation engineer there could be a managed by relationship in two instances of the same class.

### Multiplicity



is the active logical association when the cardinality of a class in relation to another is being depicted. For example, one fleet may include multiple airplanes, while one commercial airplane may contain zero to many passengers. The notation 0..\* in the diagram means “zero to many”.

Aggregation



refers to the formation of a particular class because of one class being aggregated or built as a collection. For example, the class “library” is made up of one or more books, among other materials. In aggregation, the contained classes are not strongly dependent on the lifecycle of the container. In the same example, books will remain so even when the library is dissolved. To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

Composition



The composition relationship is very similar to the aggregation relationship. with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class. That is, the contained class will be obliterated when the container class is destroyed. For example, a shoulder bag’s side pocket will also cease to exist once the shoulder bag is destroyed.

To show a composition relationship in a UML diagram, use a directional line connecting the two classes, with a filled diamond shape adjacent to the container class and the directional arrow to the contained class.

Inheritance / Generalization



refers to a type of relationship wherein one associated class is a child of another by virtue of assuming the same functionalities of the parent class. In other words, the child class is a specific type of the parent class. To show inheritance in a UML diagram, a solid line from the child class to the parent class is drawn using an unfilled arrowhead.

### Realization



denotes the implementation of the functionality defined in one class by another class. To show the relationship in UML, a broken line with an unfilled solid arrowhead is drawn from the class that defines the functionality of the class that implements the function. In the example, the printing preferences that are set using the printer setup interface are being implemented by the printer.

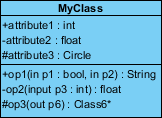
### ****Visibility in Class Diagrams****

In object-oriented design, there is a notation of visibility for attributes and operations. UML identifies four types of visibility: public, protected, private, and package.

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
* ~ denotes package attributes or operations

Class Visibility Example



In the example above:

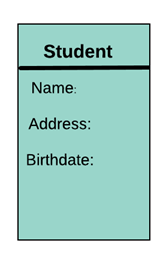
* attribute1 and op1 of MyClassName are public
* attribute3 and op3 are protected.
* attribute2 and op2 are private.

Access for each of these visibility types is shown below for members of different classes.

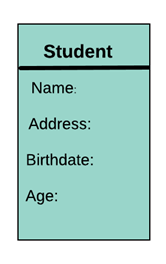
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Access Right | public (+) | private (-) | protected (#) | Package (~) |
| Members of the same class | yes | yes | yes | yes |
| Members of derived classes | yes | no | yes | yes |
| Members of any other class | yes | no | No | in same package |

### ****Attributes in Class Diagrams****

An attribute is named property of a class which describes the object being modeled. In the class diagram, this component is placed just below the name-compartment.

[](https://www.guru99.com/images/1/051818_1150_UMLClassDia2.png)

A derived attribute is computed from other attributes. For example, an age of the student can be easily computed from his/her birth date.

[](https://www.guru99.com/images/1/051818_1150_UMLClassDia3.png)

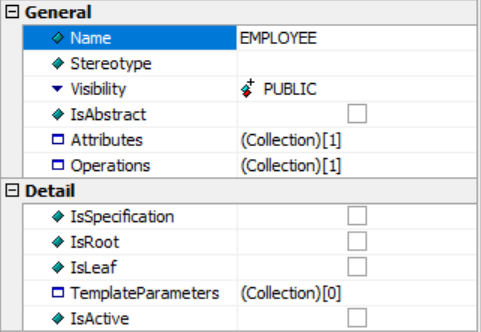
**Attributes characteristics**

* The attributes are generally written along with the visibility factor.
* Public, private, protected and package are the four visibilities which are denoted by +, -, #, or ~ signs respectively.
* Visibility describes the accessibility of an attribute of a class.
* Attributes must have a meaningful name that describes the use of it in a class.

**Example to be written in UML**

[visibility]name[':' type]['['multiplicity] ']']['=' initial-value][property-string {',', property string}]

### ****Class Diagrams Properties****



* **General Section**

1. **Name:** name of the class diagram in the UML
2. **Stereotype**: It is a profile class, which is used when we want to extend any meta class. It cannot be used by itself but should always be used with the meta class it extends. It can be used when the actor is interacting with the class or when any class wants to include or extend their functionality.
3. **Visibility**: It is the mode through which we come to know whether a class is visible or not, i.e., whether a class is accessible to another class or not.

A class diagram has 4 visibility modes:

1. **Public:** All other classes can use the classifier.
2. **Private:** Only the class itself can use the feature.
3. **Protected:** Only the classes that inherit this class can use this feature.
4. **Package:** Only classes declared in the same package can use this feature.
5. **IsAbstract**: It means that the class is incomplete and cannot be instantiated. It is just like abstract classes, which cannot be instantiated.
6. **Attributes**: It is the set of properties that a class has.
7. **Operations**: It is the set of tasks that a class performs. It is the set of functionalities, which may or may not be accessible to other classes.

* **Detail Section**

1. **IsSpecification:** Mentioning that the selected class is a specification for any other classifier.
2. **IsRoot:** Mentioning whether the selected class is a root node for any other classifier.
3. **IsLeaf:** Mentioning if the class is a leaf node for any other classifier or not.
4. **TemplateParameter:** It is like formal parameter which also can pass parameters to the class with default values.
5. **IsActive:** To check whether the class is active or not.