UNIT 2

Identification of Classes and Relationships, Identifying State and Behavior

IDENTIFICATION OF CLASSES AND RELATIONSHIPS

Class

A class represents a collection of objects having the same characteristic properties that exhibit common behavior. It gives the blueprint or description of the objects that can be created from it. Creation of an object as a member of a class is called instantiation. Thus, an object is an instance of a class.

The constituents of a class are −

- A set of attributes for the objects that are to be instantiated from the class. Generally, different objects of a class have some difference in the values of the attributes. Attributes are often referred to as class data.
- A set of operations that portray the behavior of the objects of the class. Operations are also referred to as functions or methods.

Example

Let us consider a simple class, Circle, that represents the geometrical figure circle in a two–dimensional space. The attributes of this class can be identified as follows −

- x–coord, to denote x–coordinate of the center
- y–coord, to denote y–coordinate of the center
- a, to denote the radius of the circle

Some of its operations can be defined as follows −

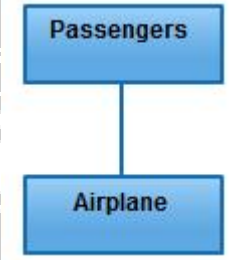
- findArea(), method to calculate area
- findCircumference(), method to calculate circumference
- scale(), method to increase or decrease the radius

Relationships in Class Diagrams

Classes are interrelated to each other in specific ways. In particular, relationships in class diagrams include different types of logical connections. The following are such types of logical connections that are possible in UML:

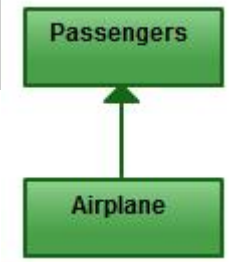
- Association
- Directed Association
- Reflexive Association
- Multiplicity
- Aggregation
- Composition
- Inheritance/Generalization
- Realization

Association



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

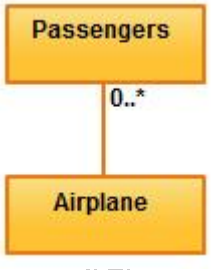


Directed Association refers to a directional relationship represented by a line with an arrowhead. The arrowhead depicts a container-contained directional flow.

Reflexive Association

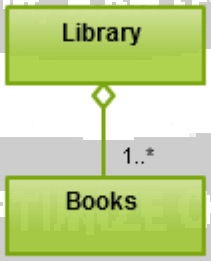
Reflexive Association occurs when a class may have multiple functions or responsibilities. For example, a staff member working in an airport may be a pilot, aviation engineer, a ticket dispatcher, a guard, or a maintenance crew member. If the maintenance crew member is managed by the aviation engineer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

Multiplicity



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

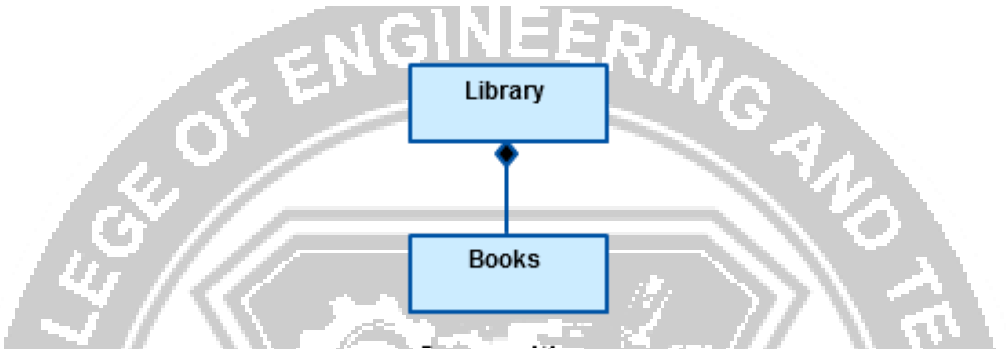


Aggregation refers to the formation of a particular class as a result of one class being aggregated or built as a collection. For example, the class “library” is made up of one or more books, among other materials. In aggregation, the contained classes are not strongly dependent on the lifecycle of the 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.

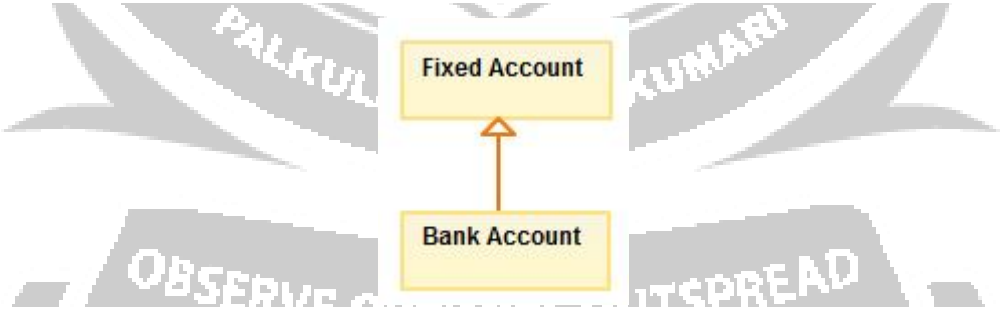
To show aggregation in a diagram, draw a line from the parent class to the child class with a diamond shape near the parent class.

Composition



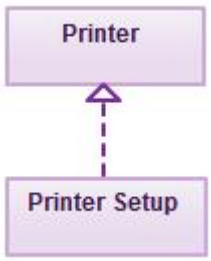
The composition relationship is very similar to the aggregation relationship. with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class. That is, the contained class will be obliterated when the container class is destroyed. 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



Inheritance 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



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.

IDENTIFYING STATE AND BEHAVIOR

State

All objects have three essential features:

- state
- behavior
- identity

An object's state is defined by the attributes of the object and by the values these have.
An attribute is a feature of an object, which distinguishes it from other kinds of objects. For example, one of the attributes of any car object is that it is capable of movement – it has a speed.
An attribute is usually something static. This means that it cannot be removed or altered without changing the essential nature of the object to which it belongs.

For example, a car that could never be moved would not be a typical car.
Though attributes themselves are usually static, the value an attribute can have is usually dynamic.

So you can alter the state of an object by changing the value of its attributes.

Behavior and identity

You can model the states of a system by defining the states of the objects that represent it. But to capture the complexity of real world problems, you also need to consider how these objects behave. Objects always exist in some kind of relation to other objects.

Sometimes they act on other objects, or even on themselves. And sometimes they are acted on by other objects.
The ways in which an object behaves are determined by its operations. So when an object needs to act on another object – to retrieve data, for instance – it uses an operation.
An operation occurs when a message is passed between two objects, allowing some function to be performed.
For structural and security reasons, some operations and attributes in a class or object are not visible

tootherclasses.

This protects a program against malicious damage. And it prevents data being accidentally deleted or overwritten by other parts of the computer program.
The hidden parts of a class can be accessed only indirectly, through the visible parts of the class.

Together, the visible operations and attributes of a class make up its interface. This is the part of a class that is acted on by operations from other classes or system users.

The three most common types of operations that a class can have are

- modifiers
- selectors
- iterators

modifiers

A modifier operation alters the state of an object.

selectors

A selector operation lets you access the state of an object without changing it.

iterators

An iterator operation allows all the parts of an object to be accessed in a definite order. Object-oriented programmers often create a separate class that is responsible for using the iterator function on an object.

Modifiers, selectors, and iterators are not part of any programming language, but are used to characterize the effects operations have.
Two types of operation are needed to create and destroy instances of a class. These are:

- Constructor
- Destructor

Constructor

The constructor operation creates an object and fixes its initial state.

Destructor

The destructor operation destroys an object.

Together, state and behavior define the roles that an object may play. And an object may play many roles during its lifetime.

For example, an object in the bank's Employee class could be involved with the payroll system, with the customer database, or with the command hierarchy.
The functions you require an object to perform are its responsibilities. And an object's responsibilities are fulfilled by its roles – by its state and behavior combined.

So you can capture all the meaningful functionality of an object by specifying its state and behavior. Objects are characterized by a third feature in addition to state and behavior – identity.

Even objects with the same properties and behavior have their own individual identity. For instance, two blue station wagons that were built in the same year by the same manufacturer are still separate

and unique cars.
The identity of an object is independent of its attributes or operations. So an object will retain its identity no matter what values its properties have.

You can, for example, respray your car or fit another engine, and it will still be the same car.