Names:
Vishal Salvi (2019230069)
Shivam Pawar (2019230068)
Shreyas Patel (2018130037)
Batch: C
Class: TE comps

Experiment No 4

Aim: To create Class Diagram for Mumbai Tour and guide management System.

Theory:

What is Class?

A Class is a blueprint that is used to create Object. The Class defines what object can do.

What is Class Diagram?

UML CLASS DIAGRAM gives an overview of a software system by displaying classes, attributes, operations, and their relationships. This Diagram includes the class name, attributes, and operation in separate designated compartments.

Class Diagram defines the types of objects in the system and the different types of relationships that exist among them. It gives a high-level view of an application. This modeling method can run with almost all Object-Oriented Methods. A class can refer to another class. A class can have its objects or may inherit from other classes.

Class Diagram helps construct the code for the software application development.

Benefits of Class Diagram

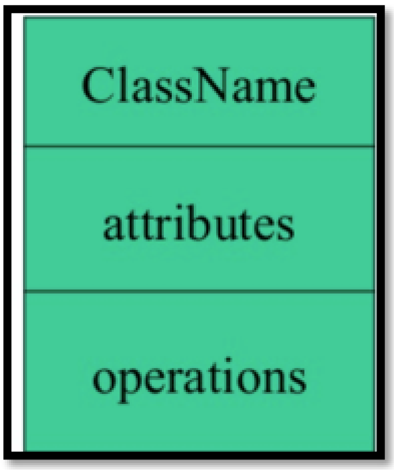
- Class Diagram Illustrates data models for even very complex information systems
- It provides an overview of how the application is structured before studying the actual code. This can easily reduce the maintenance time
- It helps for better understanding of general schematics of an application.
- Allows drawing detailed charts which highlights code required to be programmed
- Helpful for developers and other stakeholders.

Essential elements of A UML class diagram

Essential elements of UML class diagram are:

1. Class Name
2. Attributes
3. Operations

Class Name



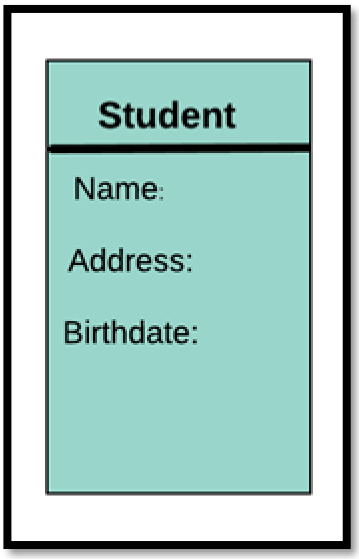
The name of the class is only needed in the graphical representation of the class. It appears in the topmost compartment. A class is the blueprint of an object which can share the same relationships, attributes, operations, & semantics. The class is rendered as a rectangle, including its name, attributes, and operations in sperate compartments.

Following rules must be taken care of while representing a class:

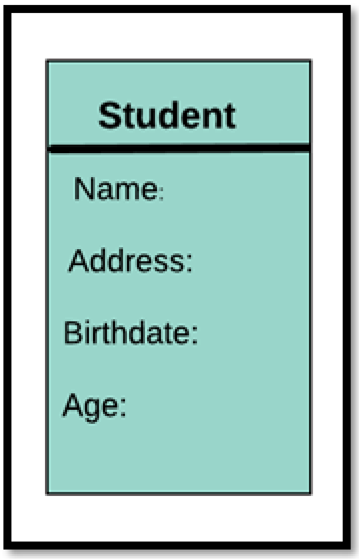
1. A class name should always start with a capital letter.
2. A class name should always be in the center of the first compartment.
3. A class name should always be written in **bold** format.
4. An abstract class name should be written in italics format.

Attributes:

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.



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



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.

Relationships

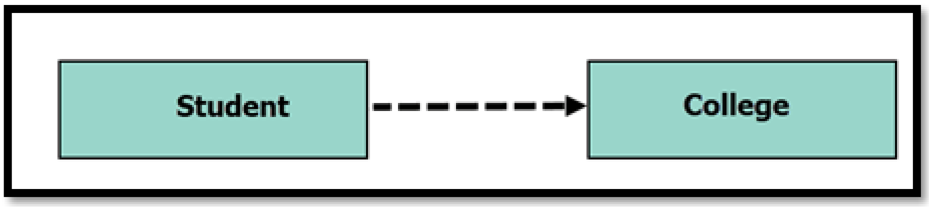
There are mainly three kinds of relationships in UML:

1. Dependencies
2. Generalizations
3. Associations

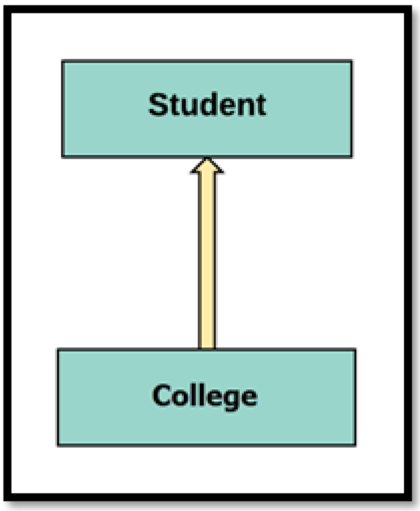
Dependency

A dependency means the relation between two or more classes in which a change in one may force changes in the other. However, it will always create a weaker relationship. Dependency indicates that one class depends on another.

In the following example, Student has a dependency on College



Generalization:



A generalization helps to connect a subclass to its superclass. A sub-class is inherited from its superclass. Generalization relationship can't be used to model interface implementation. Class diagram allows inheriting from multiple superclasses.

In this example, the class Student is generalized from Person Class.

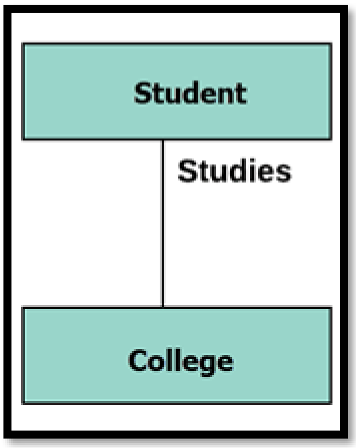
Association:

This kind of relationship represents static relationships between classes A and B. For example; an employee works for an organization.

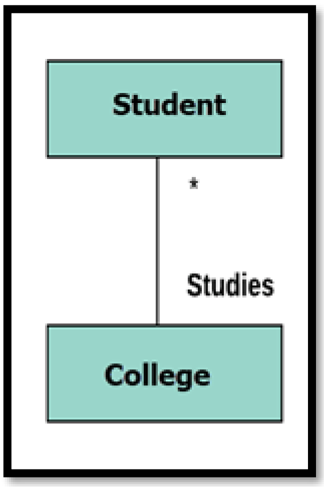
Here are some rules for Association:

- Association is mostly verb or a verb phrase or noun or noun phrase.
- It should be named to indicate the role played by the class attached at the end of the association path.
- Mandatory for reflexive associations

In this example, the relationship between student and college is shown which is studies.



Multiplicity

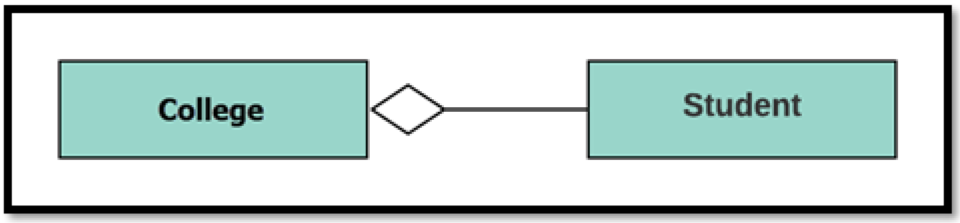


A multiplicity is a factor associated with an attribute. It specifies how many instances of attributes are created when a class is initialized. If a multiplicity is not specified, by default one is considered as a default multiplicity.

Let's say that that there are 100 students in one college. The college can have multiple students.

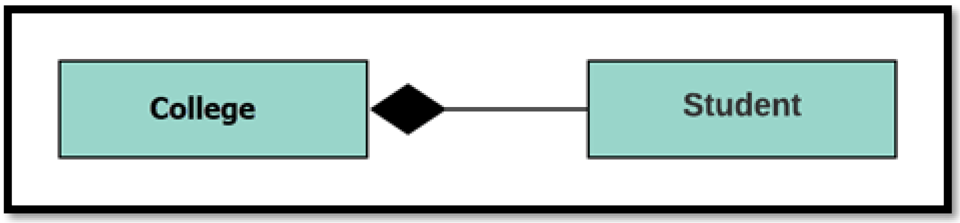
Aggregation

Aggregation is a special type of association that models a whole- part relationship between aggregate and its parts.



For example, the class college is made up of one or more student. In aggregation, the contained classes are never totally dependent on the lifecycle of the container. Here, the college class will remain even if the student is not available.

Composition:



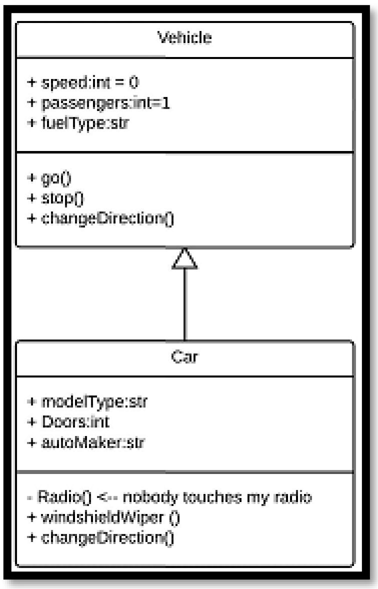
The composition is a special type of aggregation which denotes strong ownership between two classes when one class is a part of another class.

For example, if college is composed of classes student. The college could contain many students, while each student belongs to only one college. So, if college is not functioning all the students also removed.

Aggregation vs. Composition

Aggregation	**Composition**
Aggregation indicates a relationship where the child can exist separately from their parent class. Example: Automobile (Parent) and Car (Child). So, If you delete the Automobile, the child Car still exist**.**	Composition display relationship where the child will never exist independent of the parent. Example: House (parent) and Room (child). Rooms will never separate into a House.

Inheritance:
The process of a child or sub-class taking on the functionality of a parent or superclass, also known as generalization. It's symbolized with a straight connected line with a closed arrowhead pointing towards the superclass.



Class Diagram in Software Development Lifecycle

Class diagrams can be used in various software development phases. It helps in modeling class diagrams in three different perspectives.

1. Conceptual perspective: Conceptual diagrams are describing things in the real world. You should draw a diagram that represents the concepts in the domain under study. These concepts related to class and it is always language-independent.

2. Specification perspective: Specification perspective describes software abstractions or components with specifications and interfaces. However, it does not give any commitment to specific implementation.

3. Implementation perspective: This type of class diagrams is used for implementations in a specific language or application. Implementation perspective, use for software implementation.

Best practices of Designing of the Class Diagram

Class diagrams are the most important UML diagrams used for software application development. There are many properties which should be considered while drawing a Class Diagram. They represent various aspects of a software application.

Here, are some points which should be kept in mind while drawing a class diagram:

- The name given to the class diagram must be meaningful. Moreover, It should describe the real aspect of the system.
- The relationship between each element needs to be identified in advance.
- The responsibility for every class needs to be identified.
- For every class, minimum number of properties should be specified. Therefore, unwanted properties can easily make the diagram complicated.
- User notes should be included whenever you need to define some aspect of the diagram. At the end of the drawing, it must be understandable for the software development team.
- Lastly, before creating the final version, the diagram needs to be drawn on plain paper. Moreover, It should be reworked until it is ready for final submission.

Basic components of a class diagram

The standard class diagram is composed of three sections:

- **Upper section:** Contains the name of the class. This section is always required, whether you are talking about the classifier or an object.
- **Middle section:** Contains the attributes of the class. Use this section to describe the qualities of the class. This is only required when describing a specific instance of a class.
- **Bottom section:** Includes class operations (methods). Displayed in list format, each operation takes up its own line. The operations describe how a class interacts with data.

Member access modifiers

All classes have different access levels depending on the access modifier (visibility). Here are the access levels with their corresponding symbols:

- Public (+)
- Private (-)
- Protected (#)
- Package (~)
- Derived (/)
- Static (underlined)

Member scopes

There are two scopes for members: classifiers and instances.

Classifiers are static members while instances are the specific instances of the class. If you are familiar with basic OO theory, this isn't anything groundbreaking.

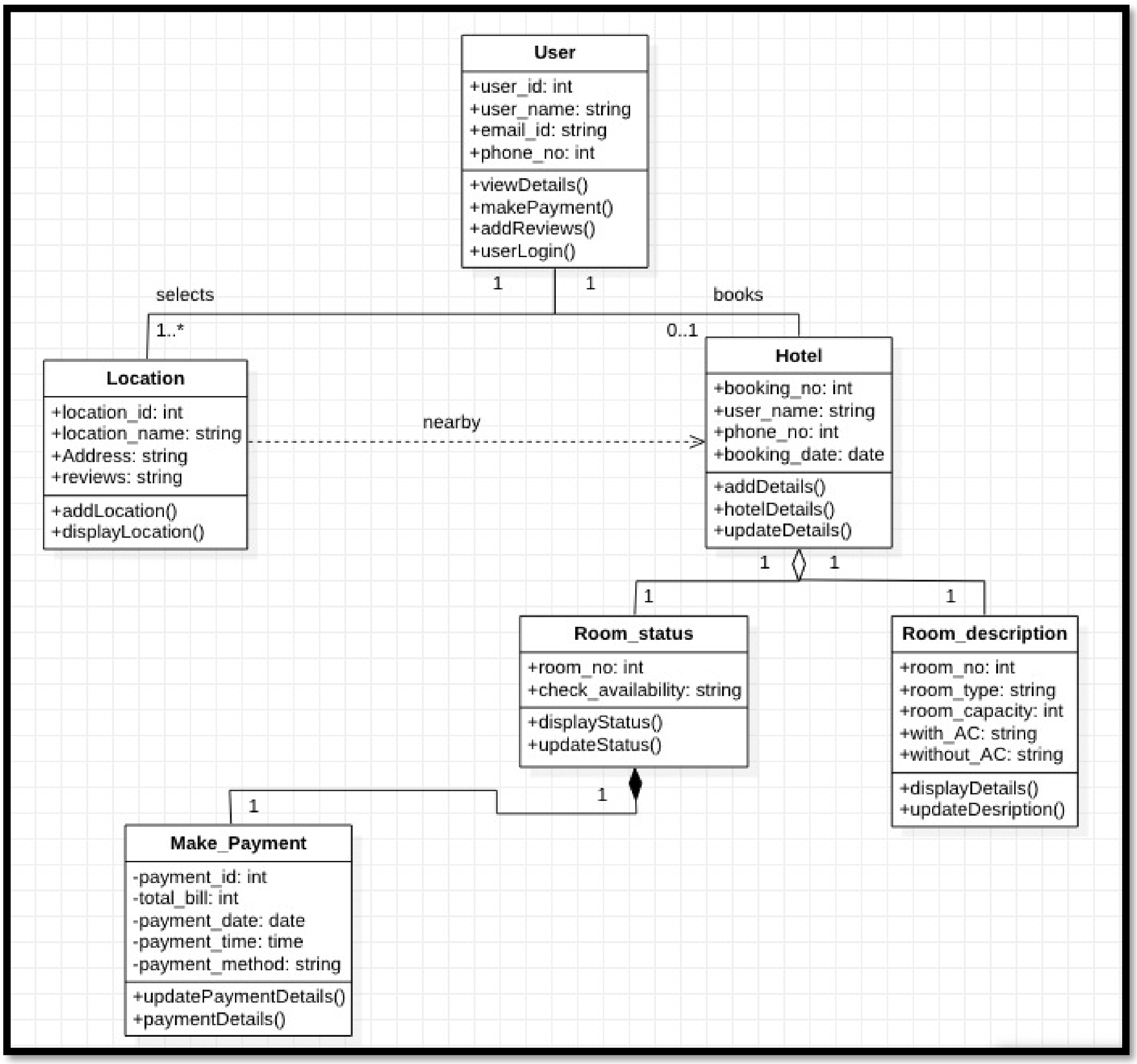
Additional class diagram components

Depending on the context, classes in a class diagram can represent the main objects, interactions in the application, or classes to be programmed. To answer the question "What is a class diagram in UML?" you should first understand its basic makeup.

- **Classes:** A template for creating objects and implementing behavior in a system. In UML, a class represents an object or a set of objects that share a common structure and behavior. They're represented by a rectangle that includes rows of the class name, its attributes, and its operations. When you draw a class in a class diagram, you're only required to fill out the top row—the others are optional if you'd like to provide more detail.

  - **Name:** The first row in a class shape.
  - **Attributes:** The second row in a class shape. Each attribute of the class is displayed on a separate line.
  - **Methods:** The third row in a class shape. Also known as operations, methods are displayed in list format with each operation on its own line.
- **Signals**: Symbols that represent one-way, asynchronous communications between active objects.
- **Data types:** Classifiers that define data values. Data types can model both primitive types and enumerations.
- **Packages:** Shapes designed to organize related classifiers in a diagram. They are symbolized with a large tabbed rectangle shape.
- **Interfaces:** A collection of operation signatures and/or attribute definitions that define a cohesive set of behaviours. Interfaces are similar to classes, except that a class can have an instance of its type, and an interface must have at least one class to implement it.
- **Enumerations:** Representations of user-defined data types. An enumeration includes groups of identifiers that represent values of the enumeration.
- **Objects:** Instances of a class or classes. Objects can be added to a class diagram to represent either concrete or prototypical instances.
- **Artifacts:** Model elements that represent the concrete entities in a software system, such as documents, databases, executable files, software components, etc.

Class Diagram:



Conclusion:

- A class diagram describes the types of objects in the system and the different kinds of relationships which exist among them.
- It allows analysis and design of the static view of a software application
- Class diagrams are most important UML diagrams used for software application development.
- Class Diagram provides an overview of how the application is structured before studying the actual code. It certainly reduces the maintenance time