Object Oriented Design: UML

Introduction

- Unified Modeling Language (UML)
- A general purpose modelling language
- The main aim of UML is to define a standard way to visualize the way a system has been designed
- It is quite similar to blueprints used in other fields of engineering
- UML is not a programming language, it is rather a visual language
- We use UML diagrams to portray the behavior and structure of a system

Why We Need UML

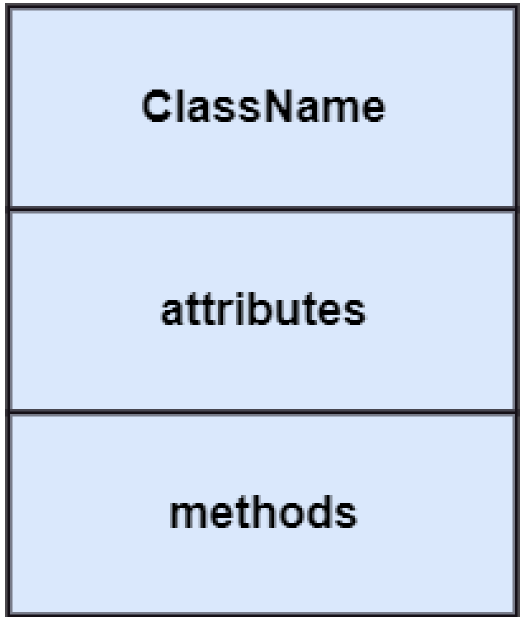
- Complex applications need collaboration and planning from multiple teams and hence require a clear and concise way to communicate amongst them
- Businessmen do not understand code. So UML becomes essential to communicate with non programmers essential requirements, functionalities and processes of the system
- A lot of time is saved down the line when teams are able to visualize processes, user interactions and static structure of the system

Class Diagram: Introduction

- This is called a **Structural UML Diagram**
- The class diagram depicts a static view of an application
- It represents the types of classes residing in the system and the relationships between them
- We use class diagrams to depict the static structure of a system by showing system’s classes, their methods and attributes.
- Class diagrams also help us identify relationship between different classes or objects.

Vital components of a Class Diagram

- The class diagram is made up of three sections:
  - Upper Section
  - Middle Section
  - Lower Section



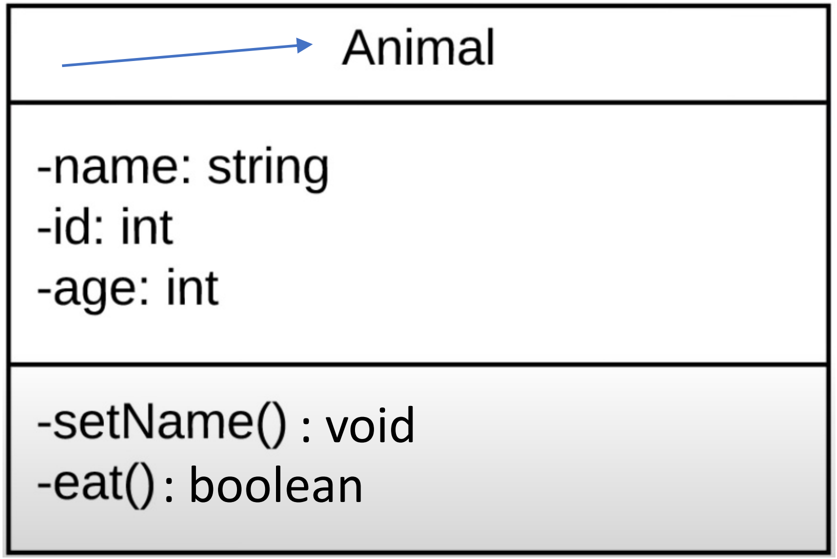
ClassName

attributes

methods

Upper Section

- The upper section encompasses the name of the class
- Some of the following rules that should be taken into account while representing a class are given below:
  - Capitalize the initial letter of the class name
  - Place the class name in the center of the upper section
  - A class name must be written in bold format
  - The name of the abstract class should be written in italics format



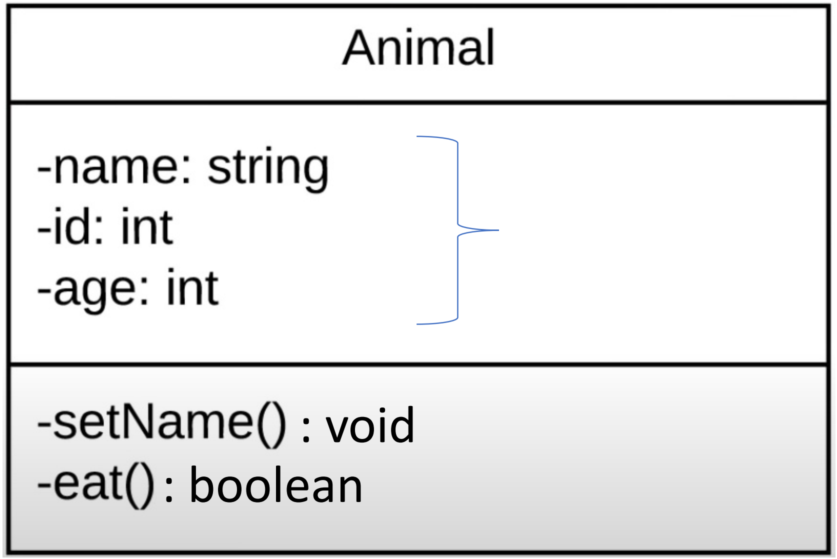
Animal

-name: string
-id: int
-age: int

-setName() : void
-eat() : boolean

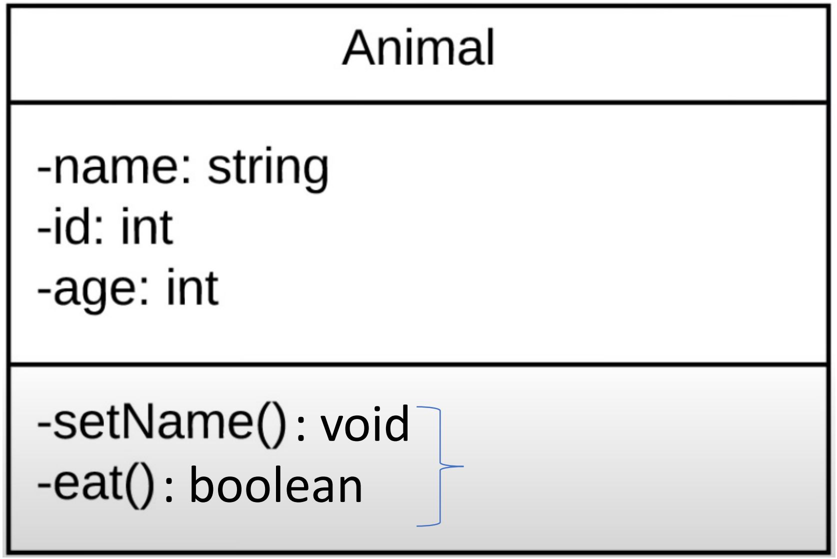
Middle Section

- The middle section constitutes the attributes, which describe the quality of the class.
- The attributes have the following characteristics:
  - The attributes are written along with its visibility factors, which are public (+), private (-), protected (#), and package (~)
  - The accessibility of an attribute class is illustrated by the visibility factors
  - A meaningful name should be assigned to the attribute, which will explain its usage inside the class
  - The type of the variables will be denoted by the name of the data type followed by a colon



Lower Section

- The lower section contain methods or operations
- Each method is written in a single line
- It demonstrates how a class interacts with data
- The methods must also have visibility factors in front of them



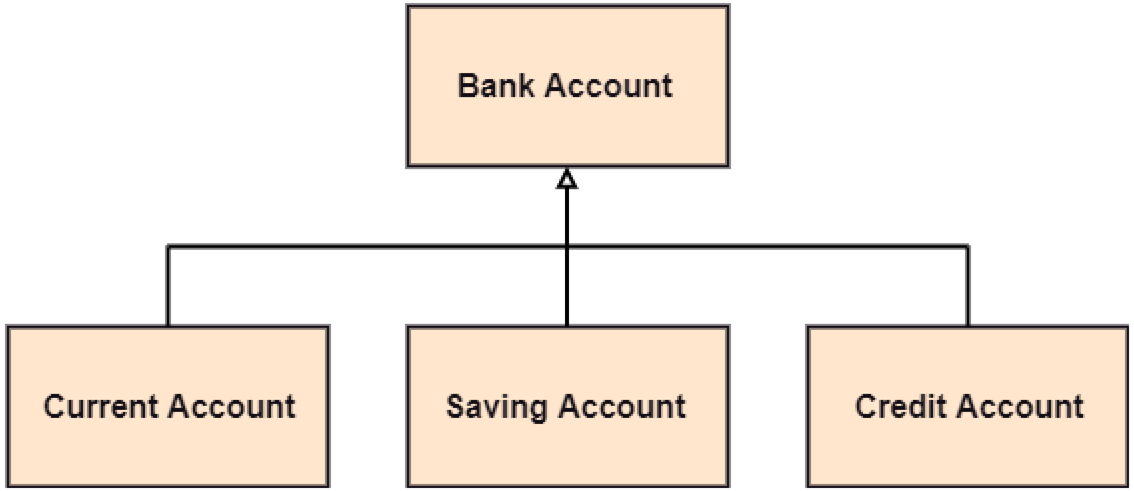
Animal

-name: string
-id: int
-age: int

-setName(): void
-eat(): boolean

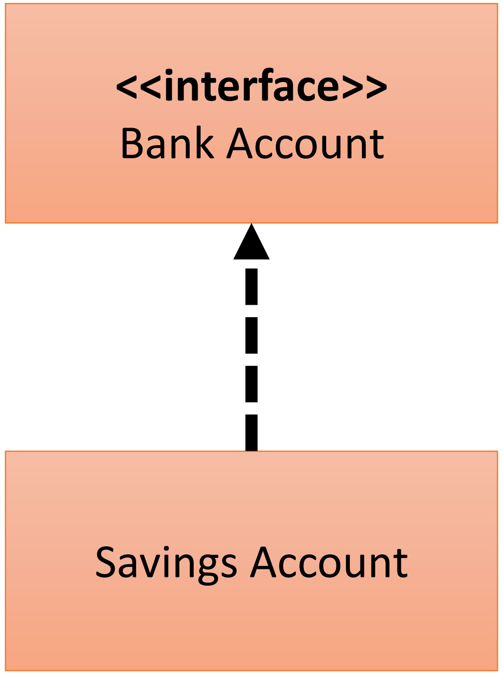
Relationships: Generalization

- A generalization is a relationship between a parent class (superclass) and a child class (subclass)
- In this, the child class is inherited from the parent class.
- For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account



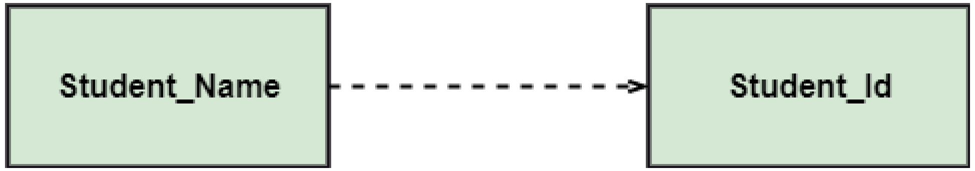
Relationships: Realization

- In a realization relationship of UML, one entity denotes some responsibility which is not implemented by itself and the other entity that implements them.
- This relationship is mostly found in the case of **interfaces.**



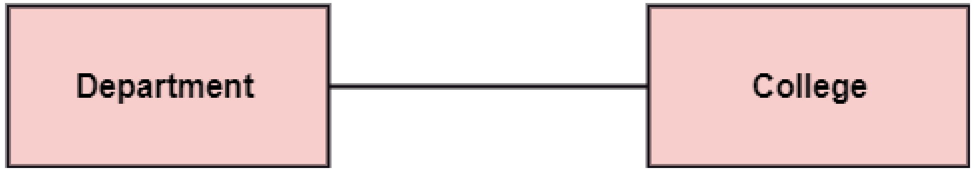
Relationships: Dependency

- A dependency is a relationship between two or more classes where a change in one class cause changes in another class
- In the following example, Student_Name is dependent on the Student_Id:



Relationships: Association

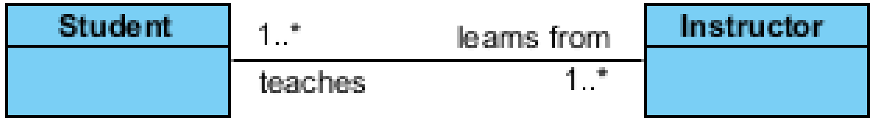
- It describes a static connection between two or more classes.
- If two classes in a model need to communicate with each other, there must be a link between them, and that can be represented by an association (connector)
- For example, a department is associated with the college



Department

College

- We can also indicate the behavior of a class in an association (i.e., the role of a class) using role names.



Student

1..*

learns from

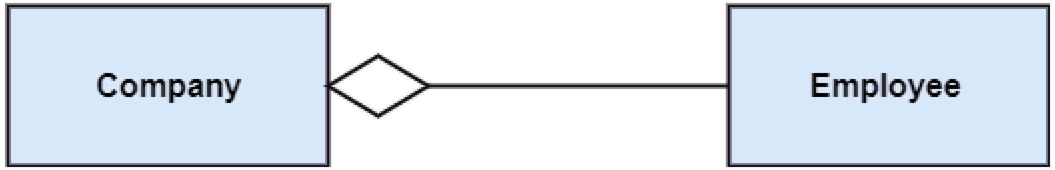
Instructor

teaches

1..*

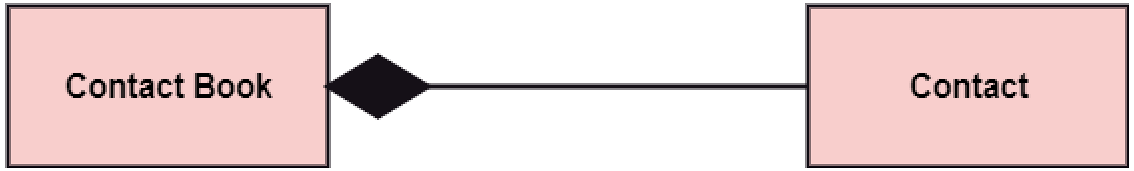
Aggregation

- Aggregation implies a relationship where the child can exist independently of the parent.
- It is more specific then association. It defines a part-of relationship.
- In this kind of relationship, the child class can exist independently of its parent class.
- The company encompasses a number of employees, even if the company gets shut down the Employee can still exist



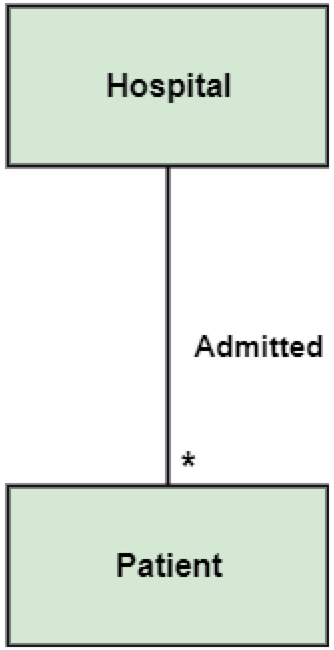
Composition

- The composition is a subset of aggregation
- It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded
- It represents a whole-part relationship
- A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost:



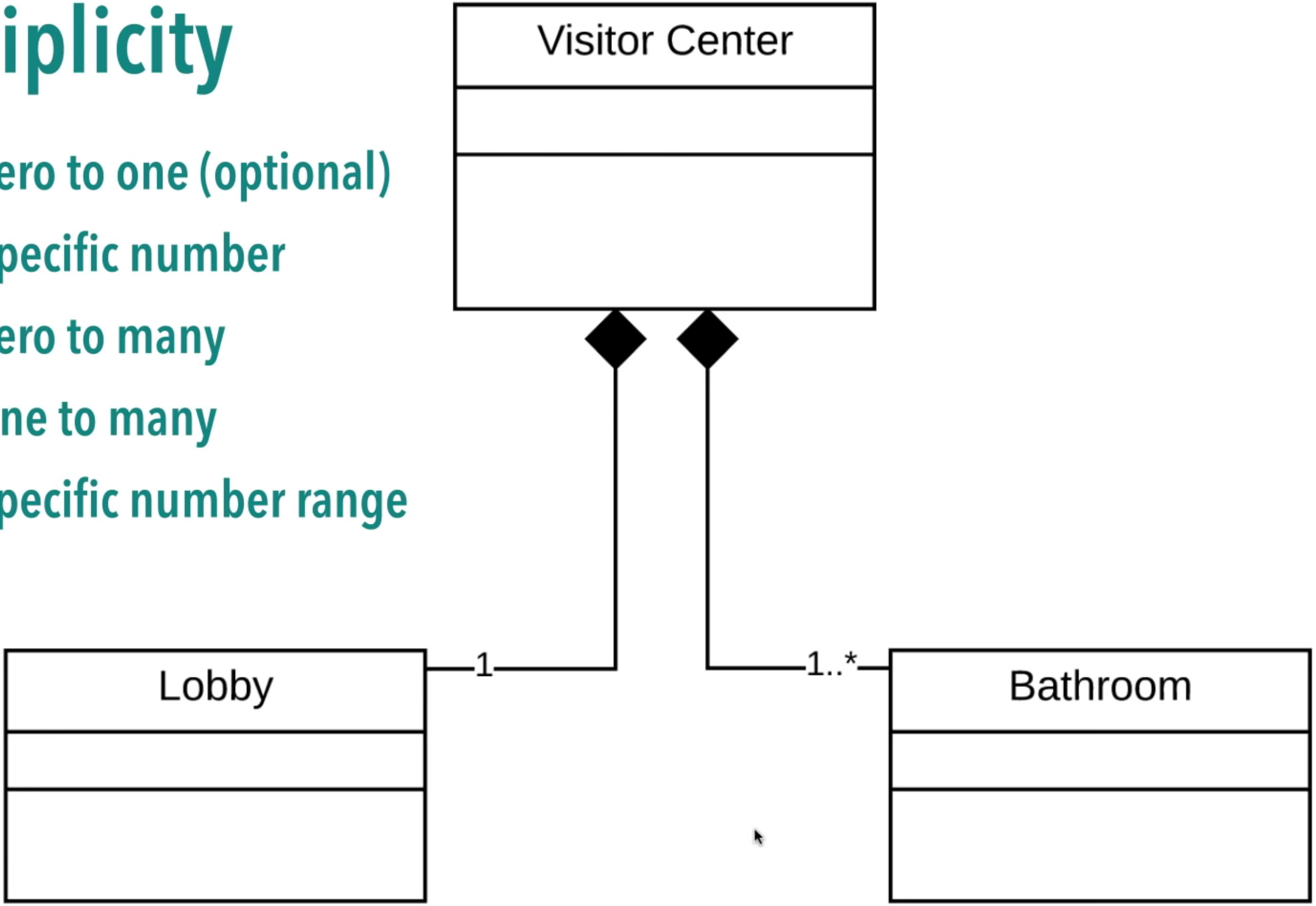
Multiplicity

- It defines a specific range of allowable instances of attributes
- In case if a range is not specified, one is considered as a default multiplicity.
- For example, multiple patients are admitted to one hospital.



Multiplicity

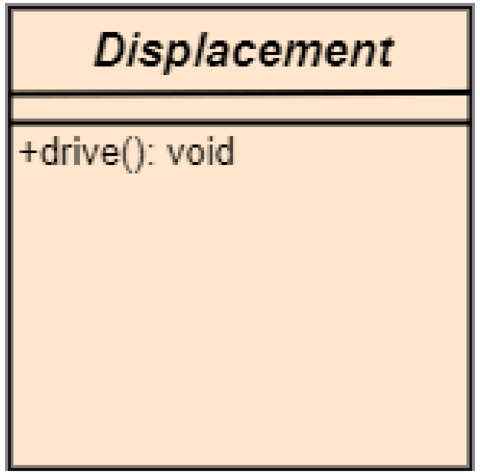
0..1	zero to one (optional)
n	specific number
0..*	zero to many
1..*	one to many
m..n	specific number range



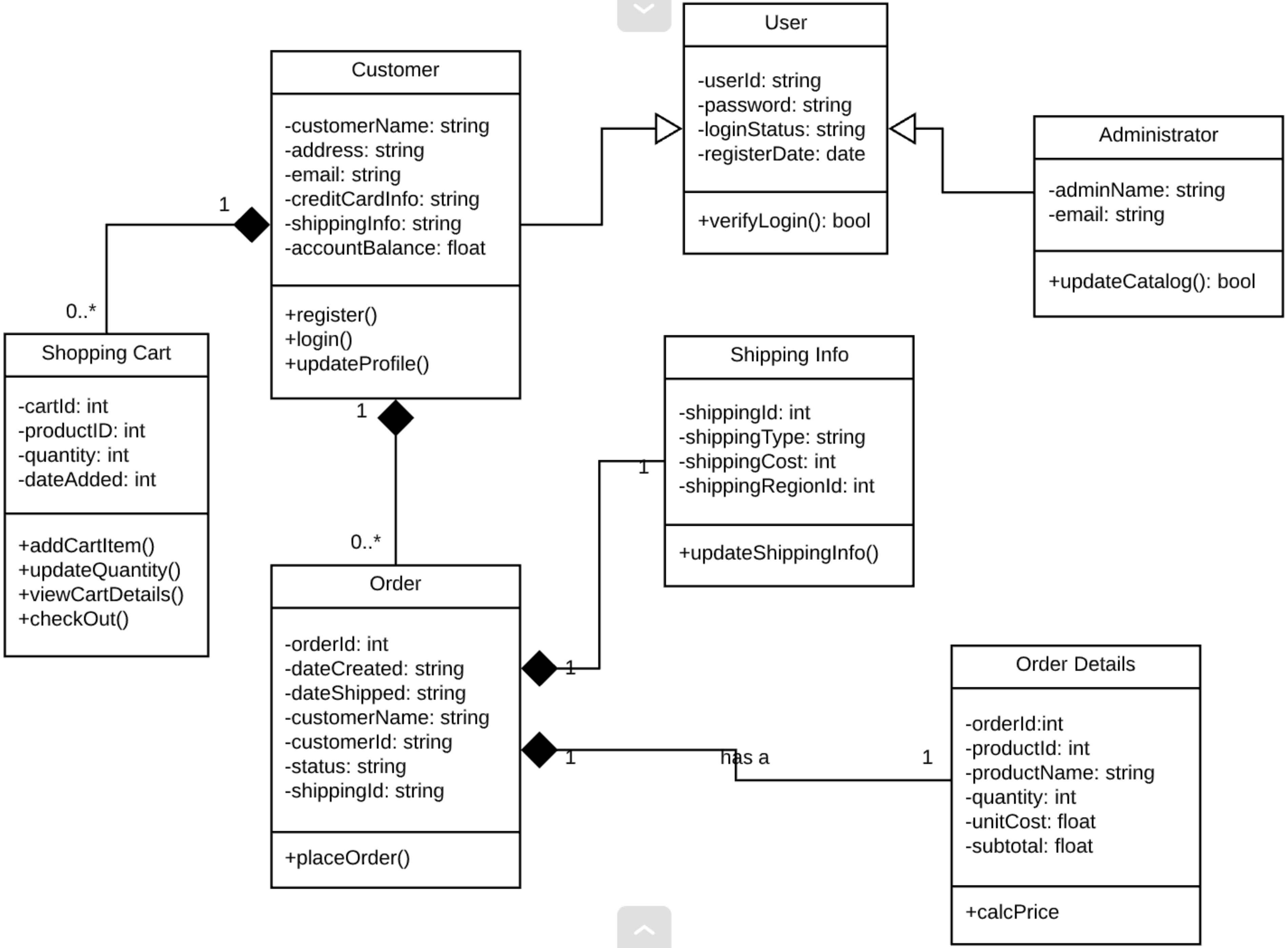
Abstract Classes

- In the abstract class, no objects can be a direct entity of the abstract class.
- The abstract class objects can not be instantiated.
- The notation of the abstract class is similar to that of class; the only difference is that the name of the class is written in italics.

- Let us assume that we have an abstract class named **displacement** with a method declared inside it, and that method will be called as a drive (). Now, this abstract class method can be implemented by any class, for example, Car, Bike, Scooter, Cycle, etc.



Class Diagram Example: Shopping Cart



User

-userId: string
-password: string
-loginStatus: string
-registerDate: date

+verifyLogin(): bool

Customer

-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float

+register()
+login()
+updateProfile()

Administrator

-adminName: string
-email: string

+updateCatalog(): bool

Shopping Cart

-cartId: int
-productID: int
-quantity: int
-dateAdded: int

+addCartItem()
+updateQuantity()
+viewCartDetails()
+checkOut()

Shipping Info

-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int

+updateShippingInfo()

Order

-orderId: int
-dateCreated: string
-dateShipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string

+placeOrder()

Order Details

-orderId:int
-productId: int
-productName: string
-quantity: int
-unitCost: float
-subtotal: float

+calcPrice