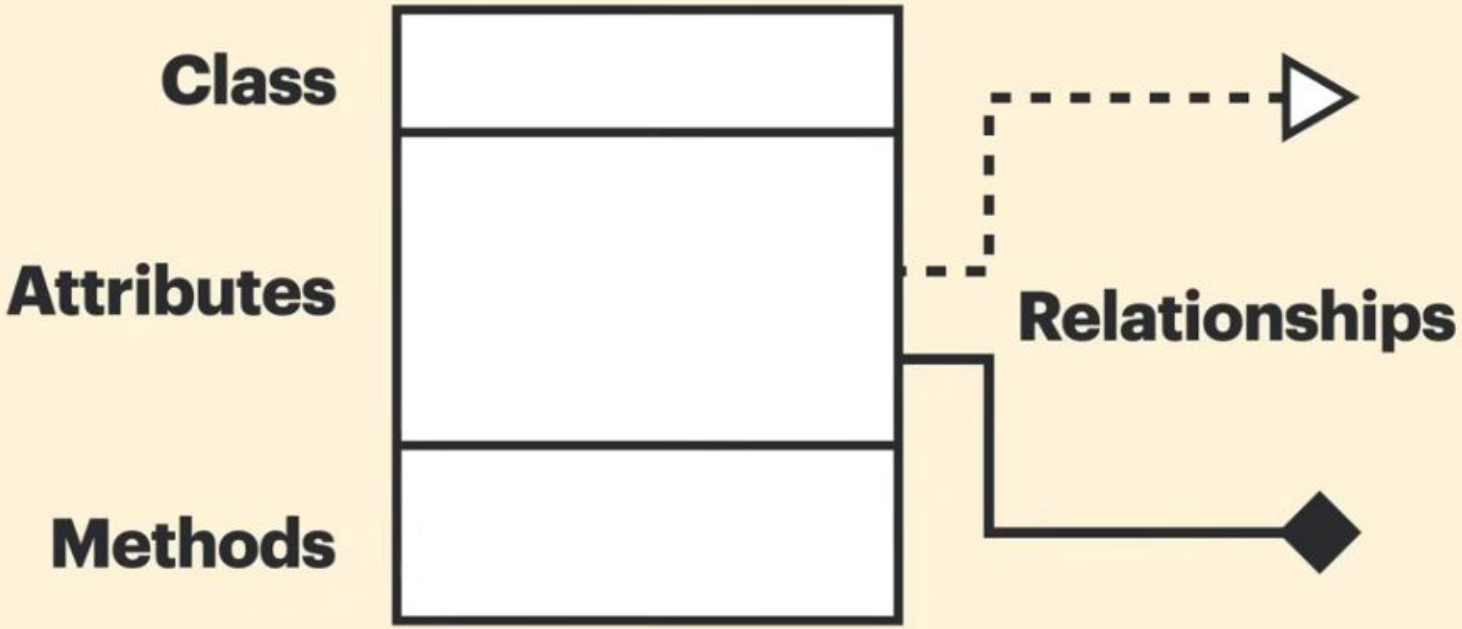
UML Class Diagrams

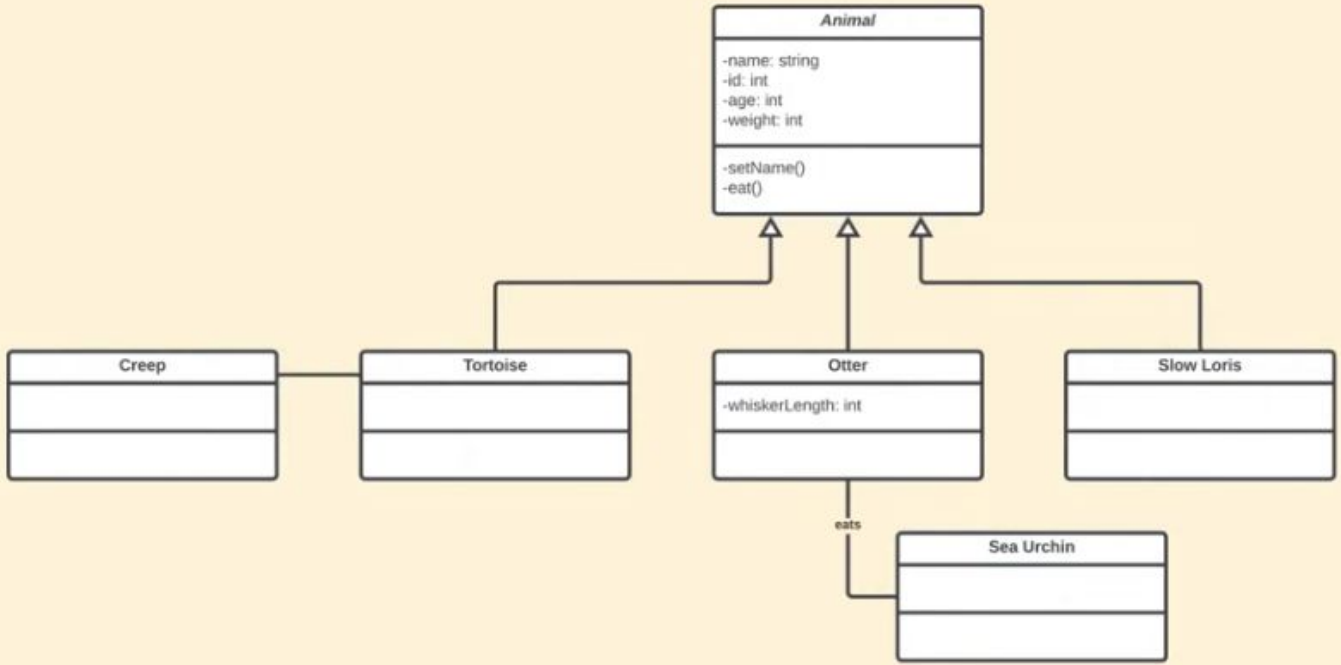
Instructor: Paruj Ratanaworabhan

Source:
Lucid Software
www.youtube.com/watch?v=6XrL5jXmTwM

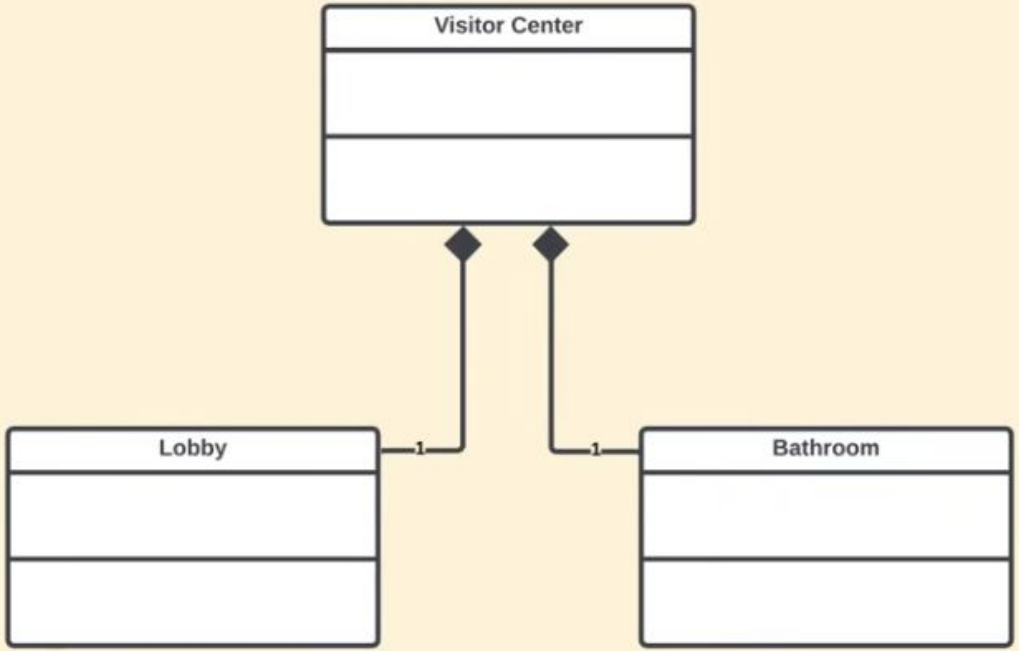
UML class diagrams



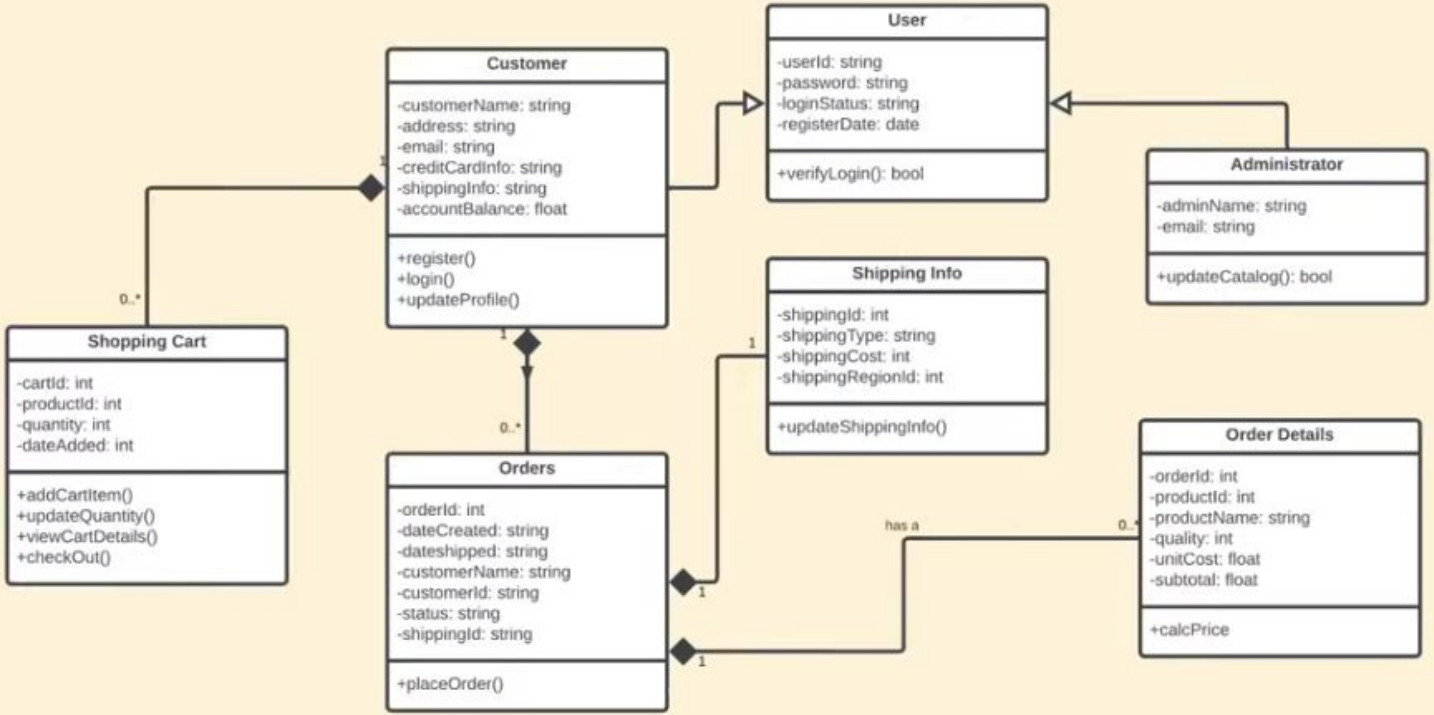
UML class diagrams



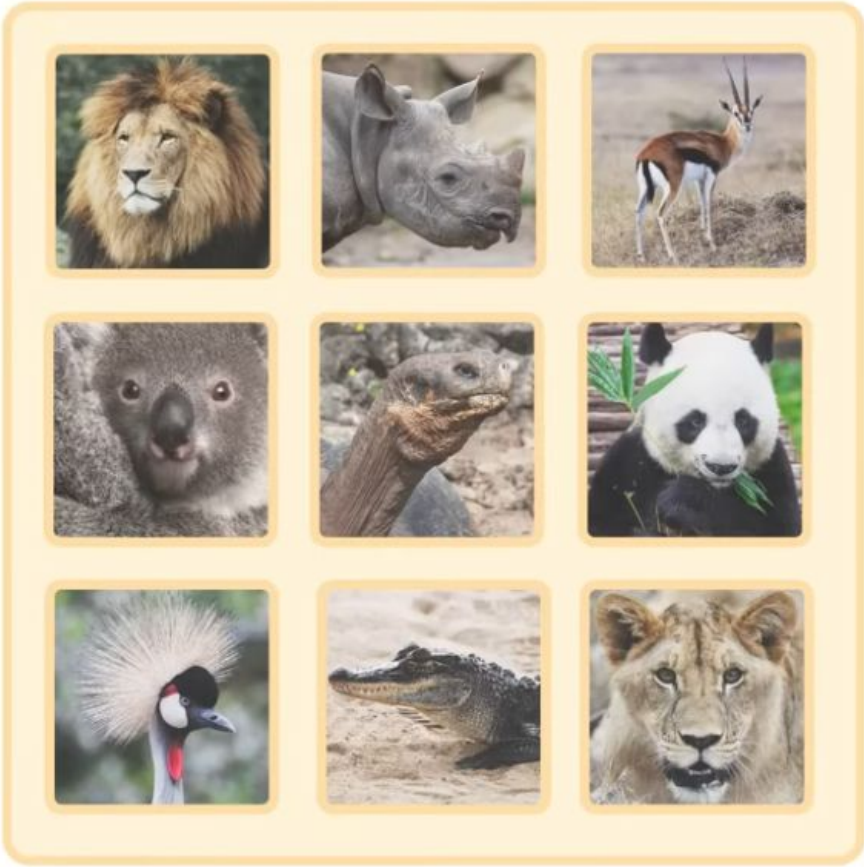
UML class diagrams



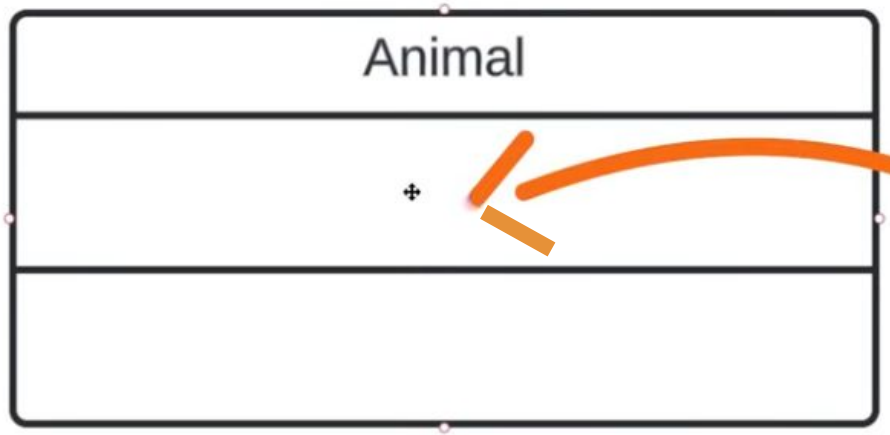
UML class diagrams



Class



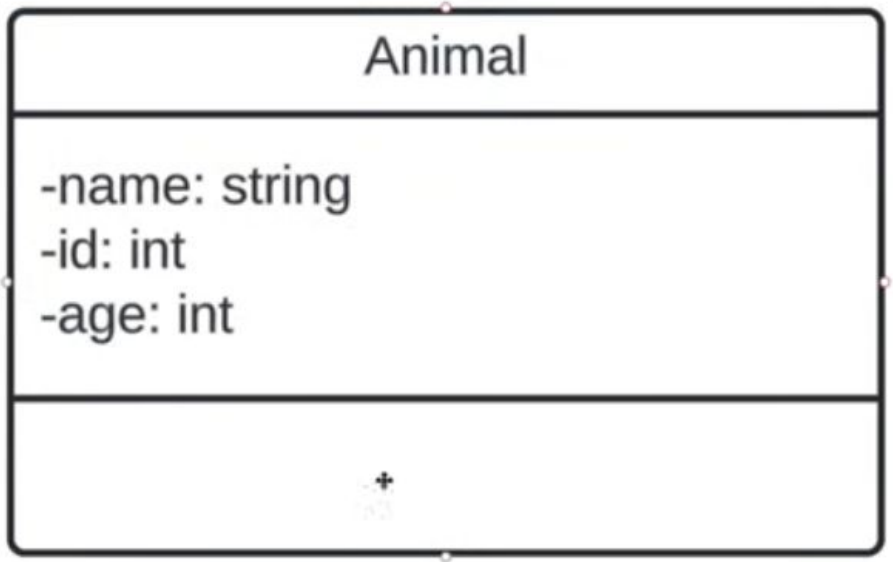
Animal

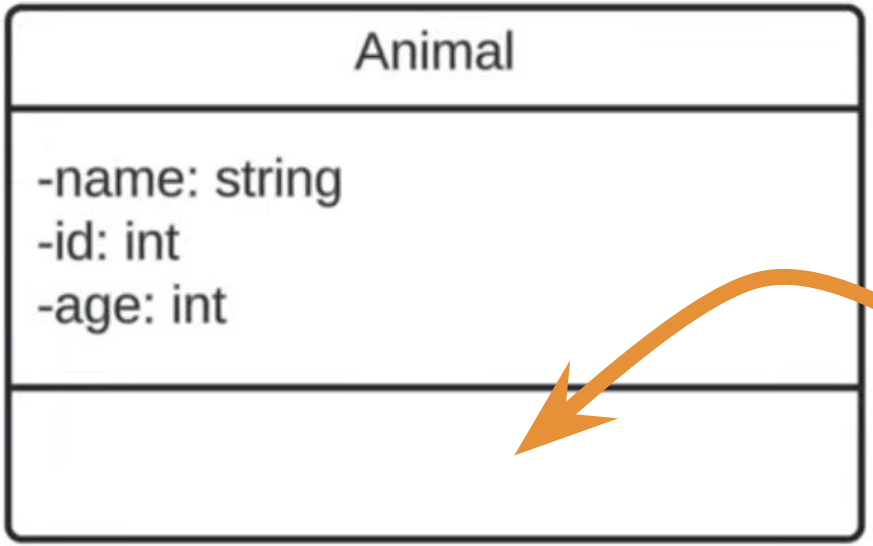


Attributes

A significant piece of data containing values that describe each instance of that class.

Also known as fields, variables, or properties.

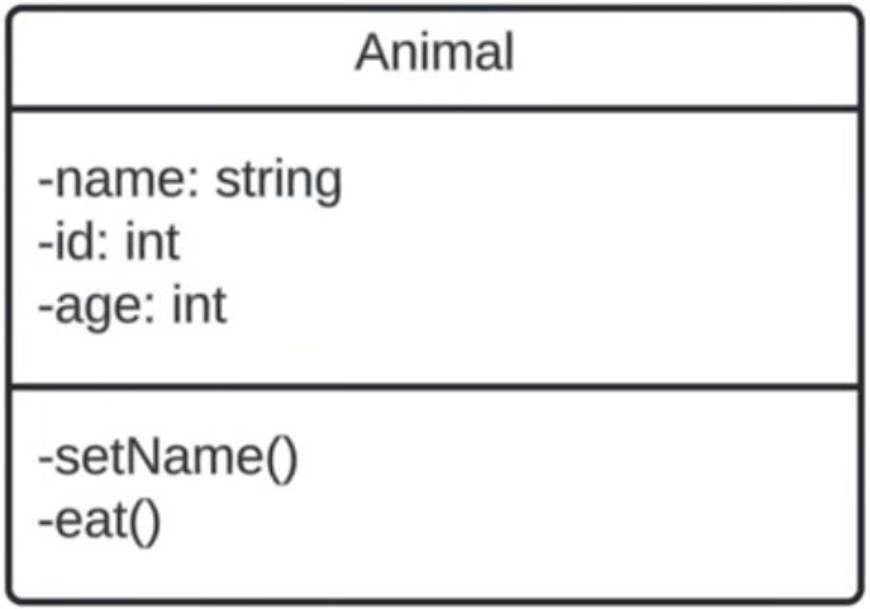

Animal
-name: string
-id: int
-age: int
+

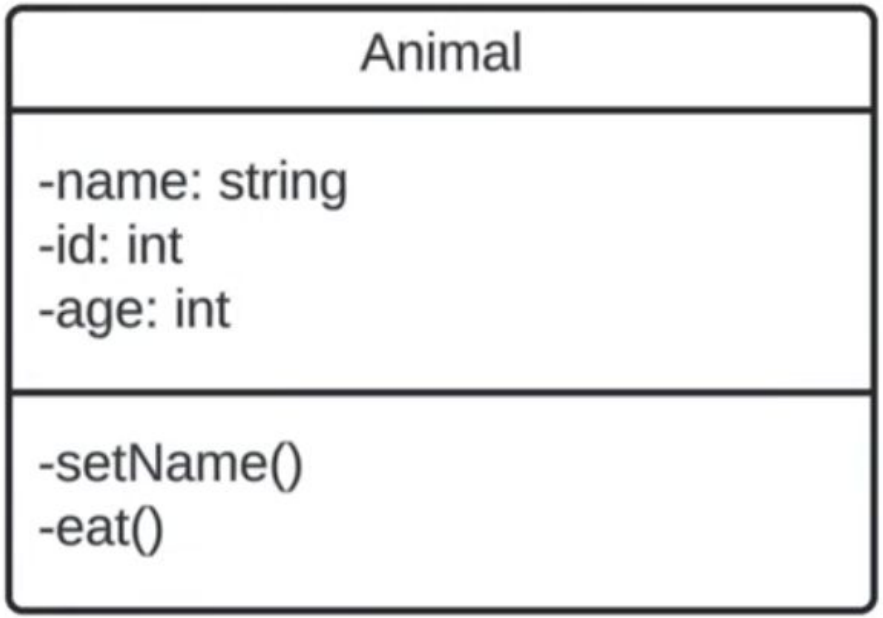


Methods

Also known as operations or functions.

Allow you to specify any behavioral features of a class.

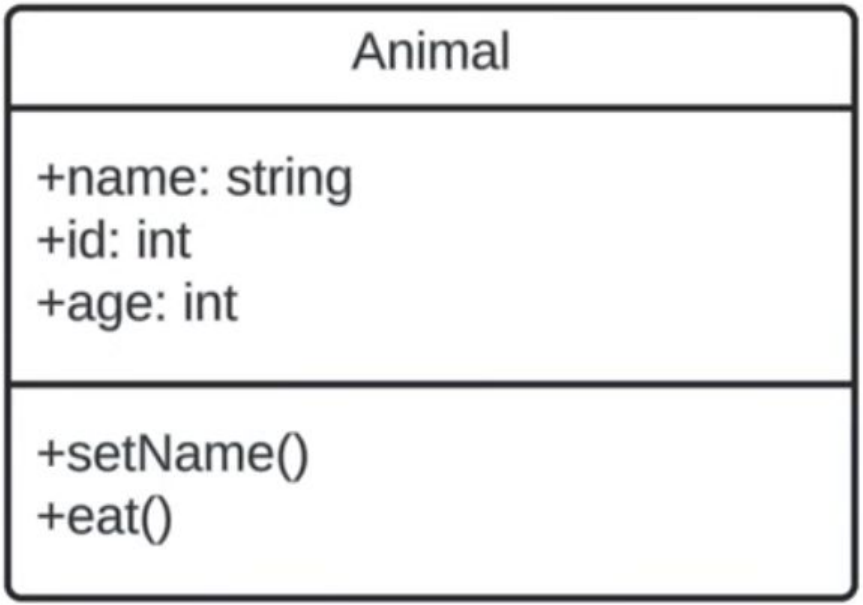

Animal
-name: string
-id: int
-age: int
-setName()
-eat()



Visibility

Sets the accessibility for that attribute or method.

- private

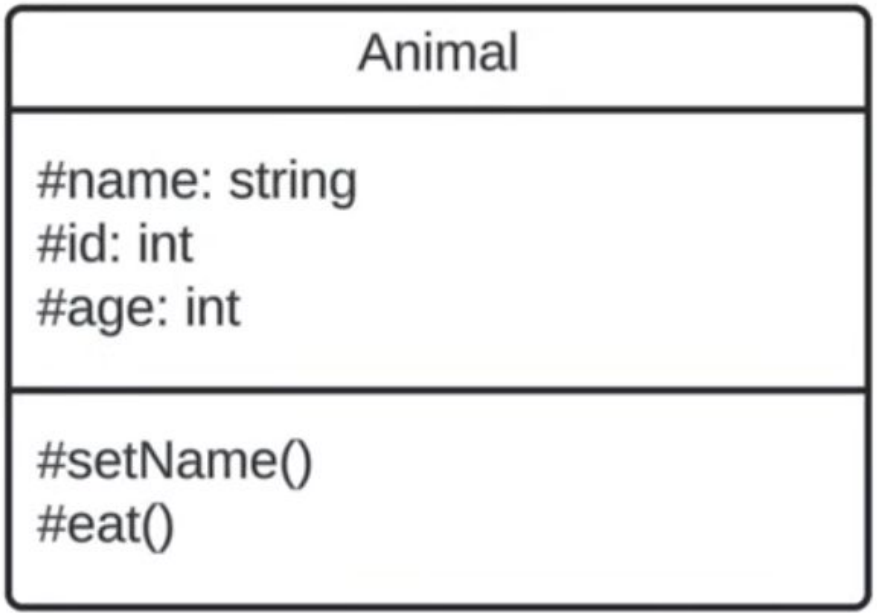


Visibility

Sets the accessibility for that attribute or method.

- private

+ public



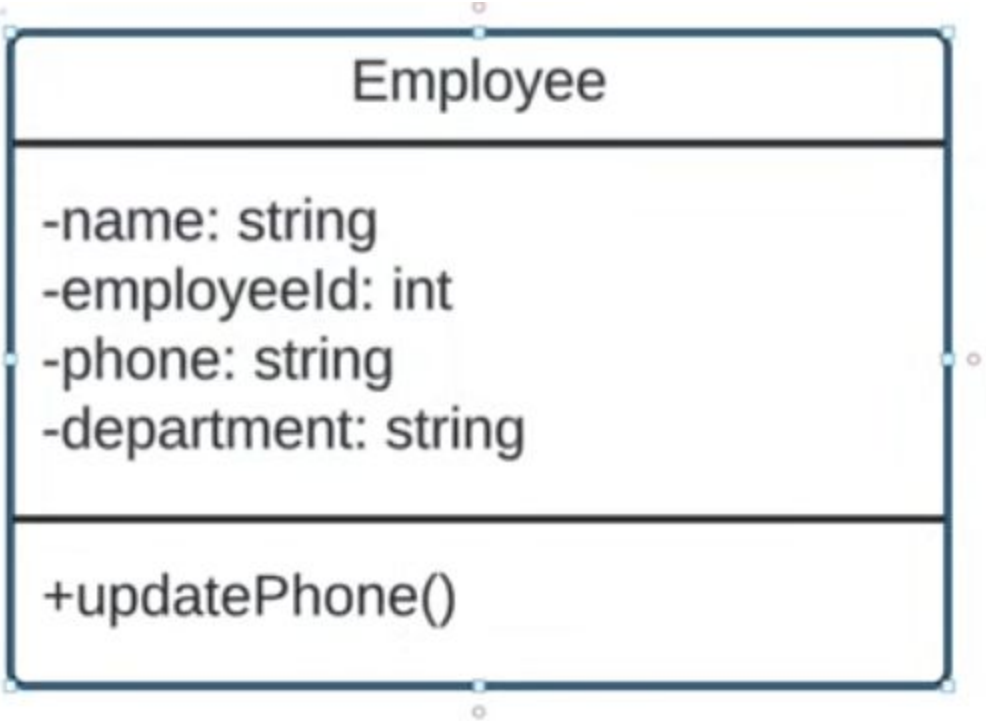
Visibility

Sets the accessibility for that attribute or method.

- private

+ public

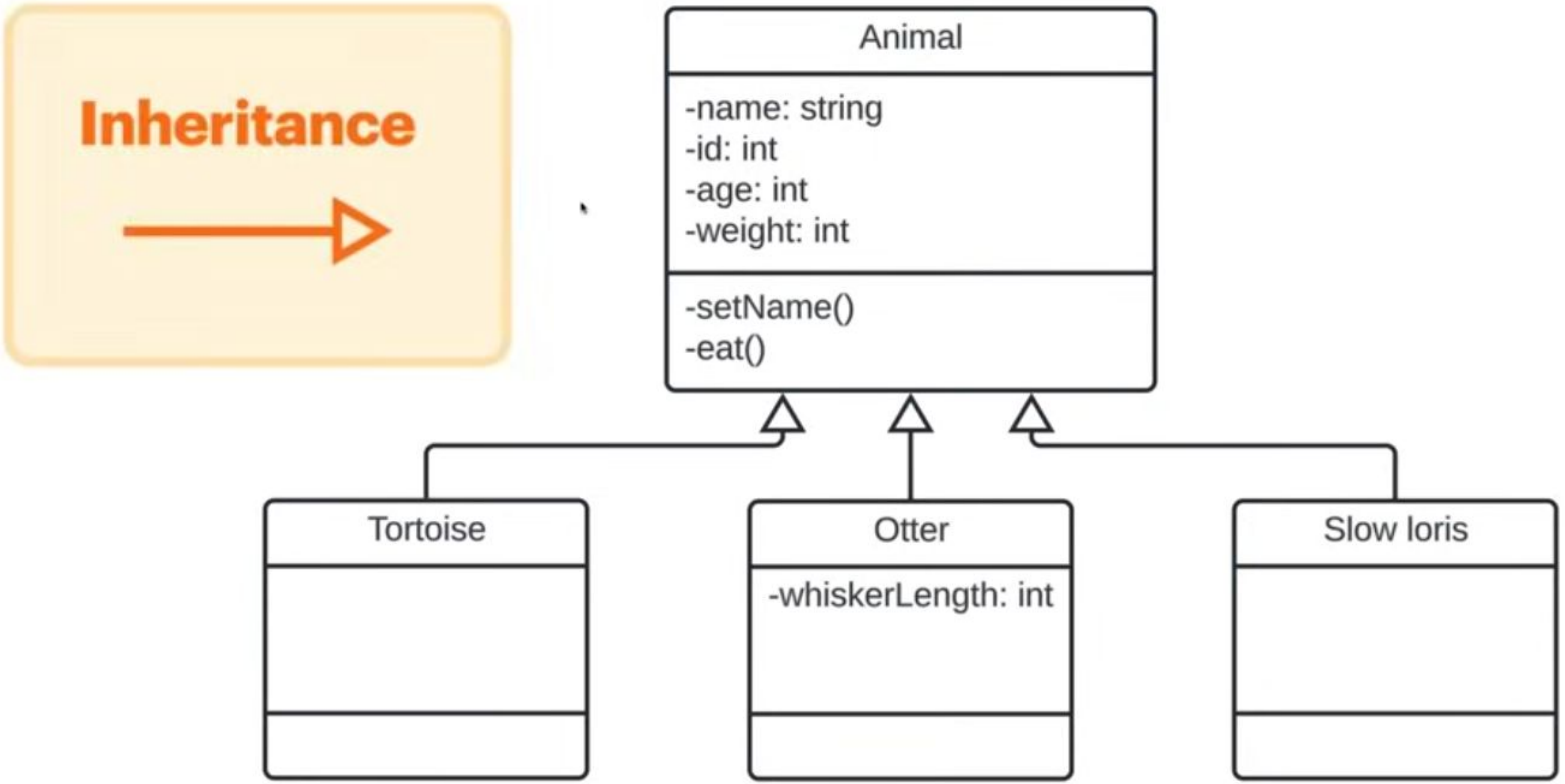
 # protected



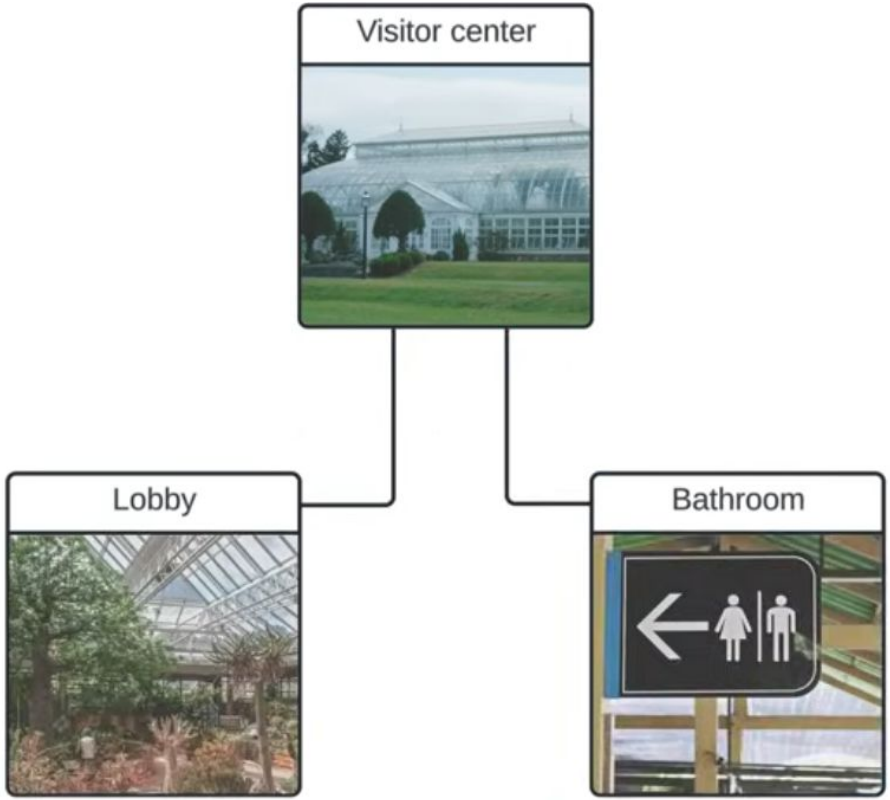
Good practice:
Attribute visibility = private

Method (accessible to the outside world) visibility = public

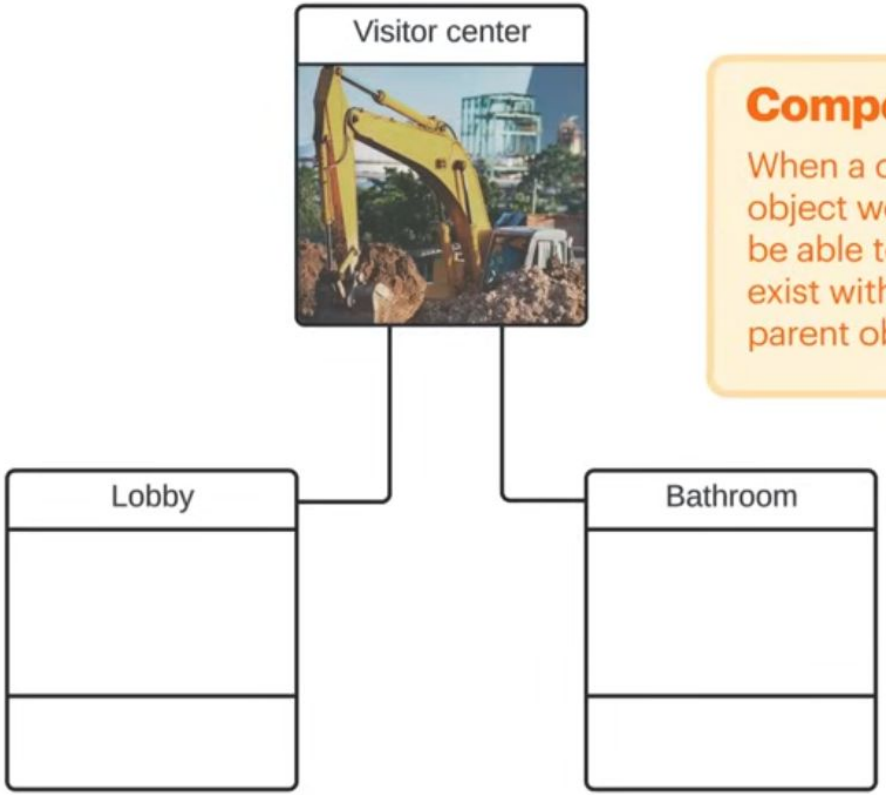
Relationships: IS-A



Relationships: HAS-A



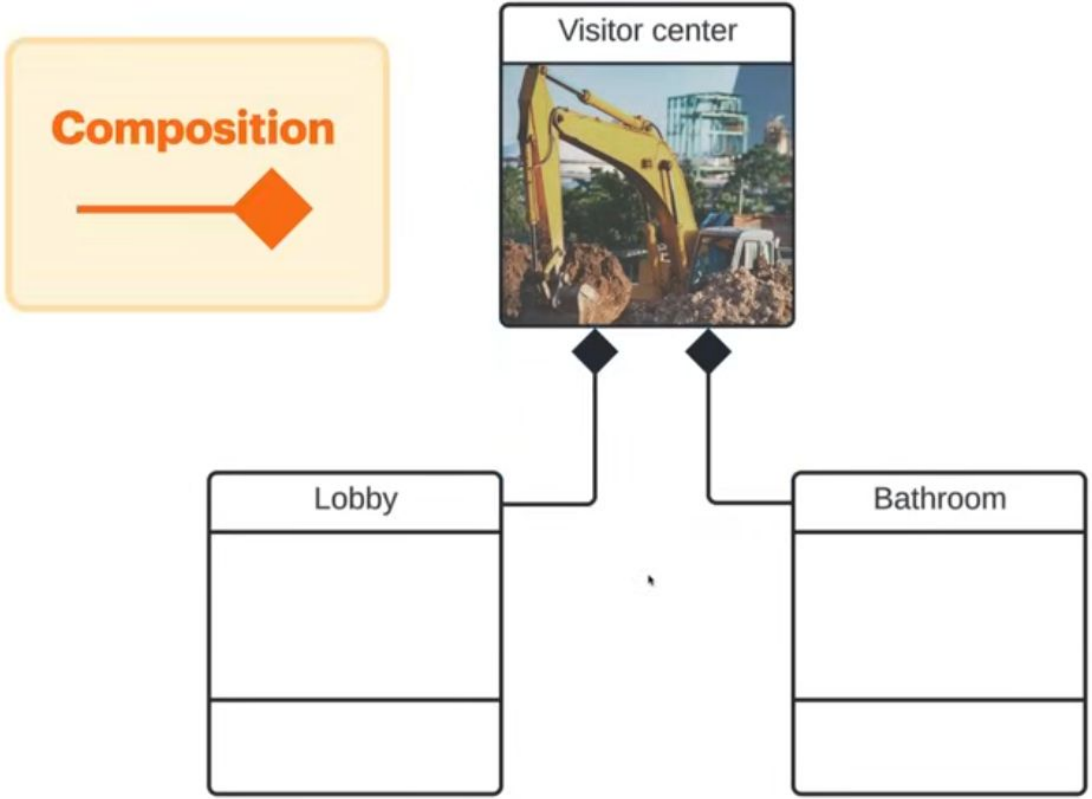
Relationships: HAS-A



Composition

When a child object wouldn't be able to exist without its parent object.

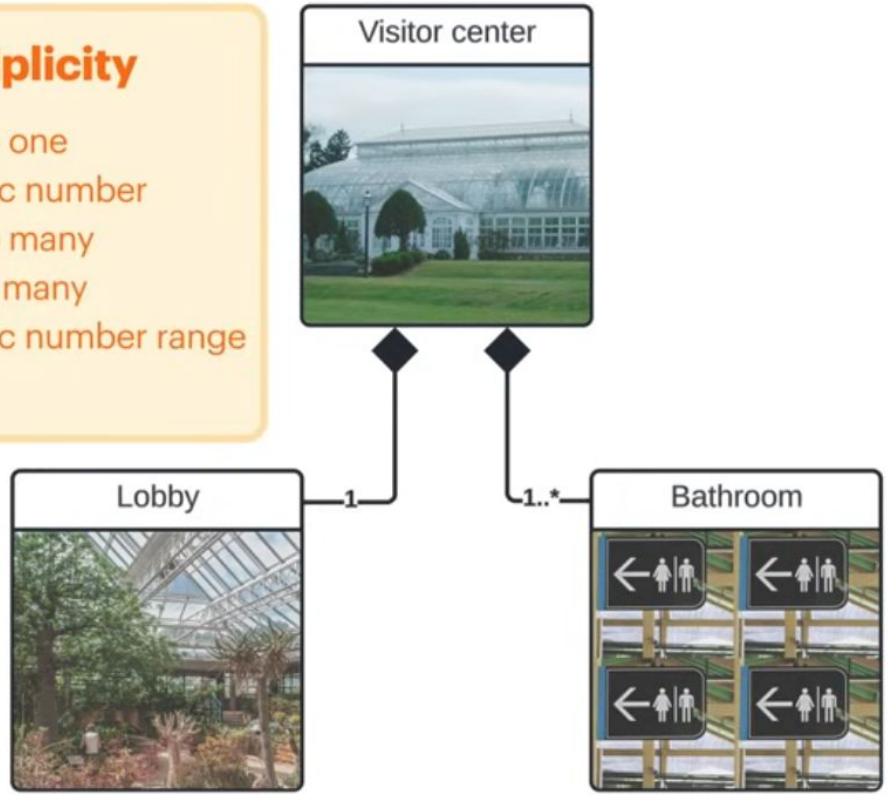
Relationships: HAS-A



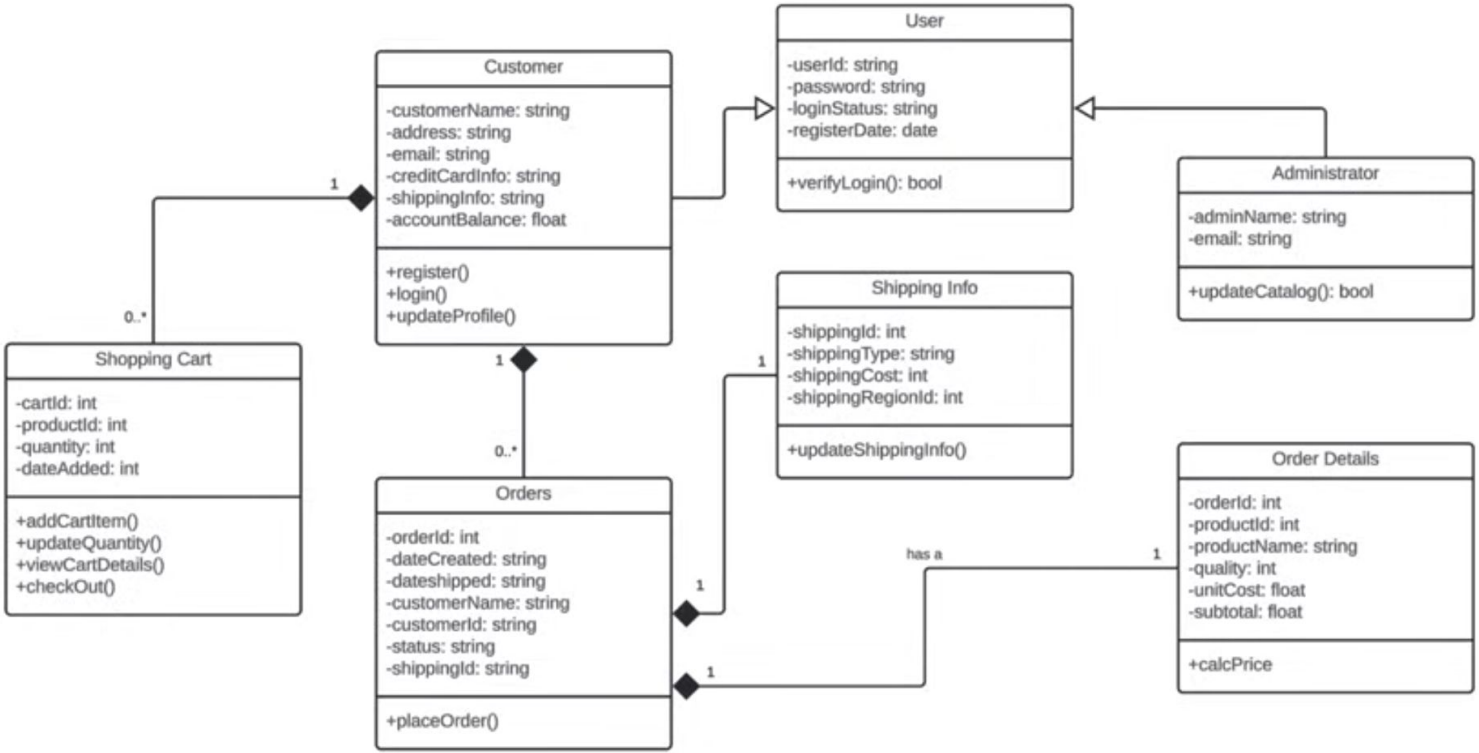
Relationships: HAS-A

Multiplicity

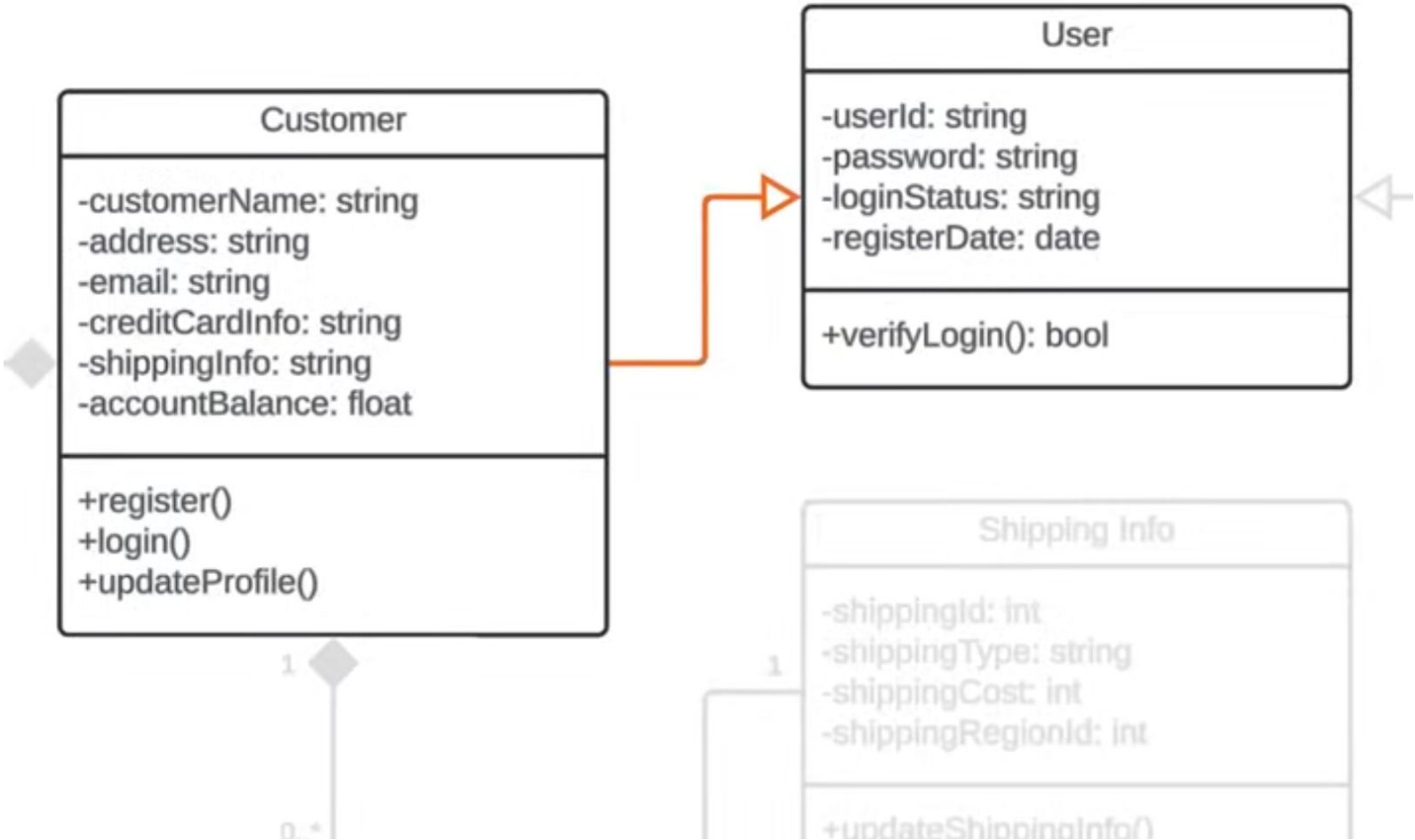
0..1	Zero to one
n	Specific number
0..*	Zero to many
1..*	One to many
m..n	Specific number range

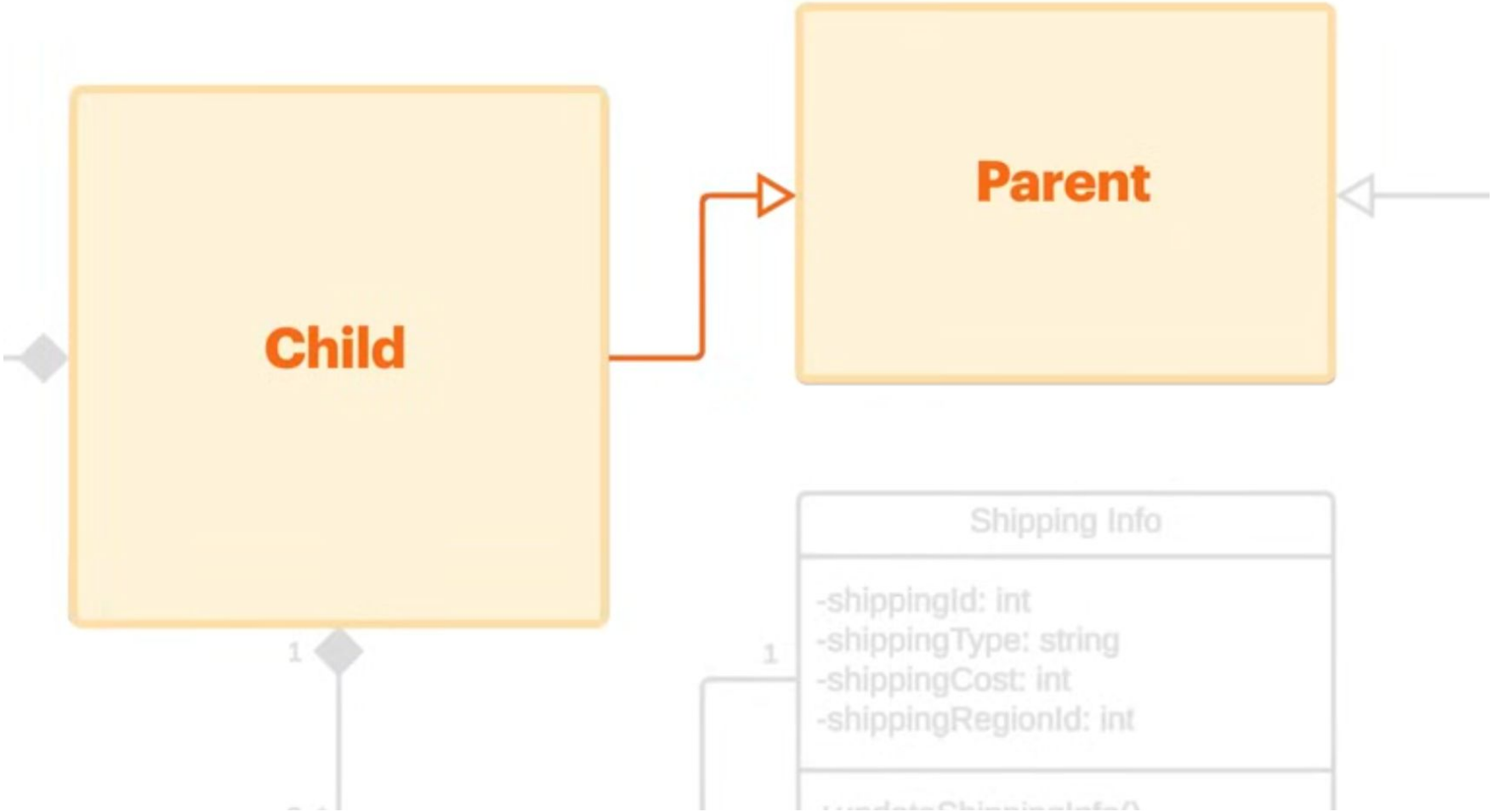


A real world example

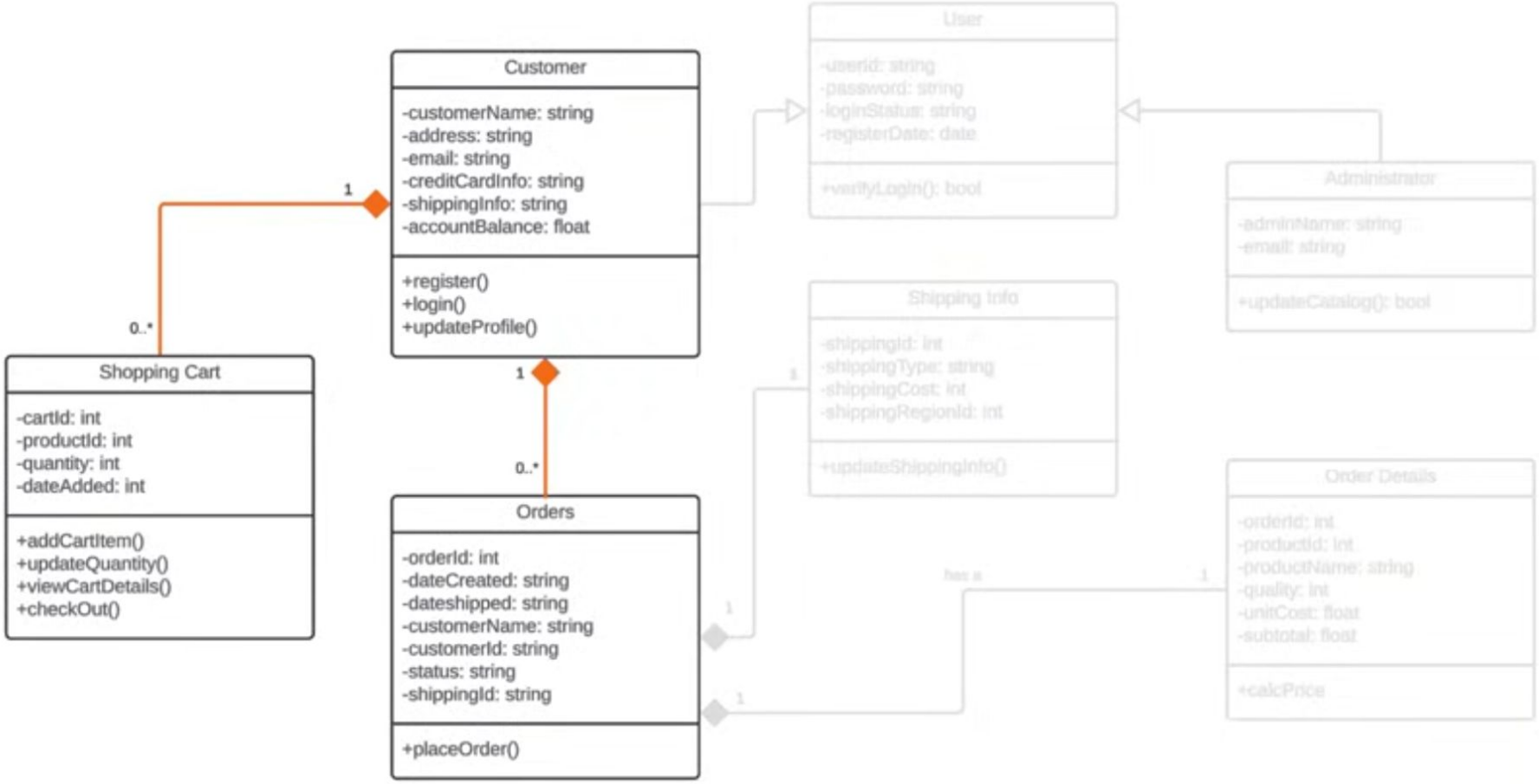


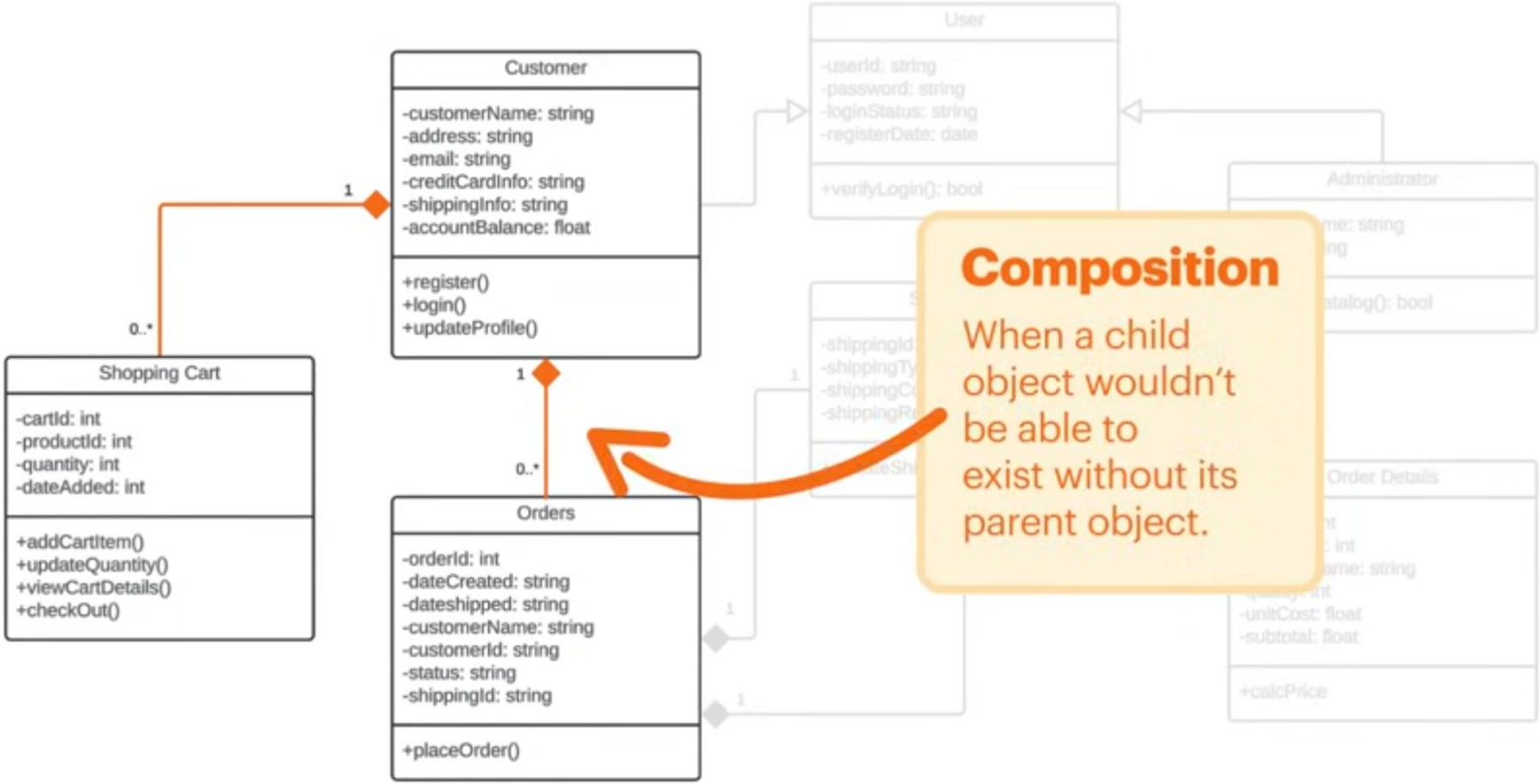
Inheritance: IS-A

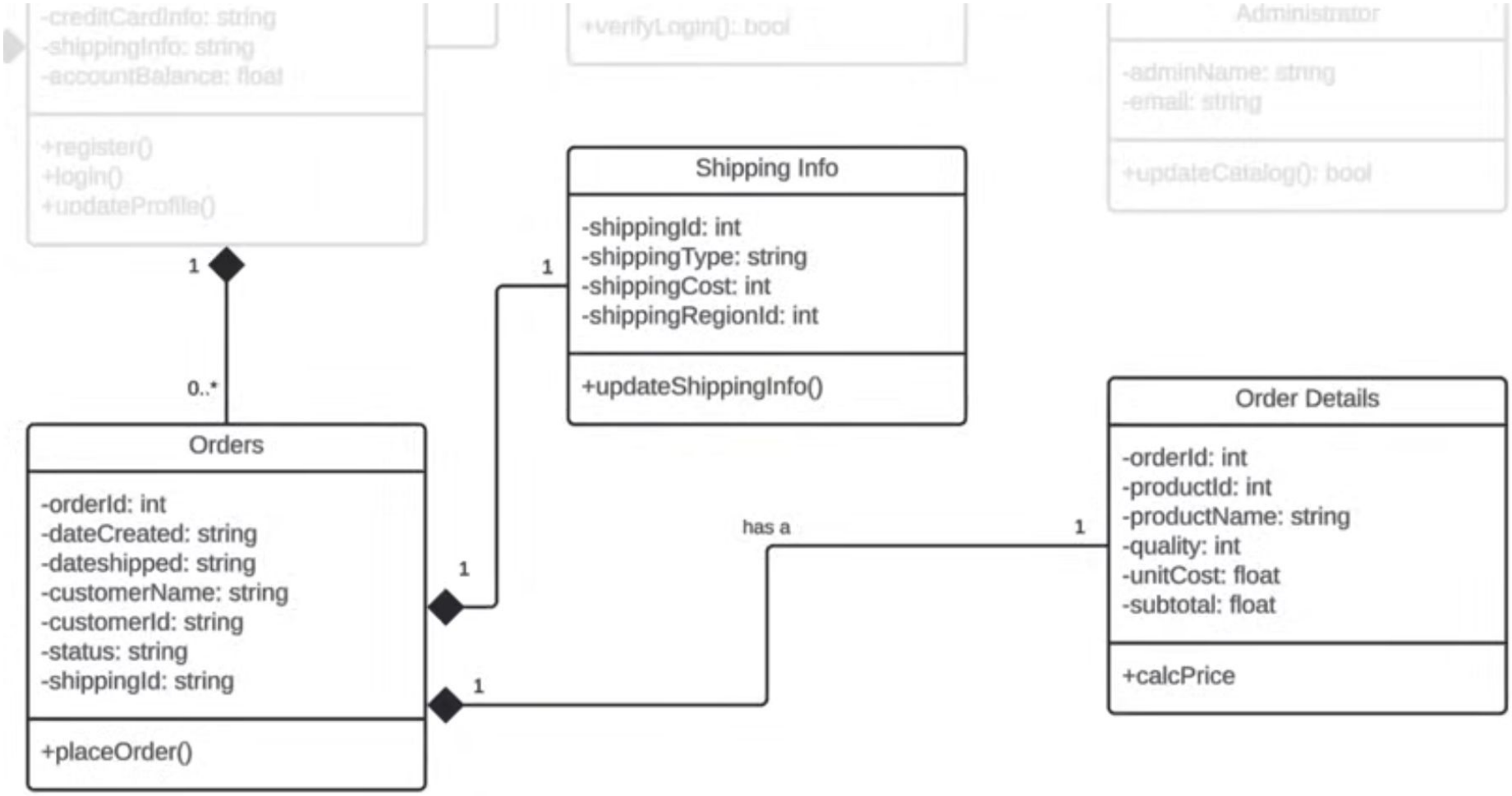

User
-userId: string
-password: string
-loginStatus: string
-registerDate: date
+verifyLogin(): bool
Customer
-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
Shipping Info
-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int
1
1
0..*


Child
Parent
Shipping Info
-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int
1
1

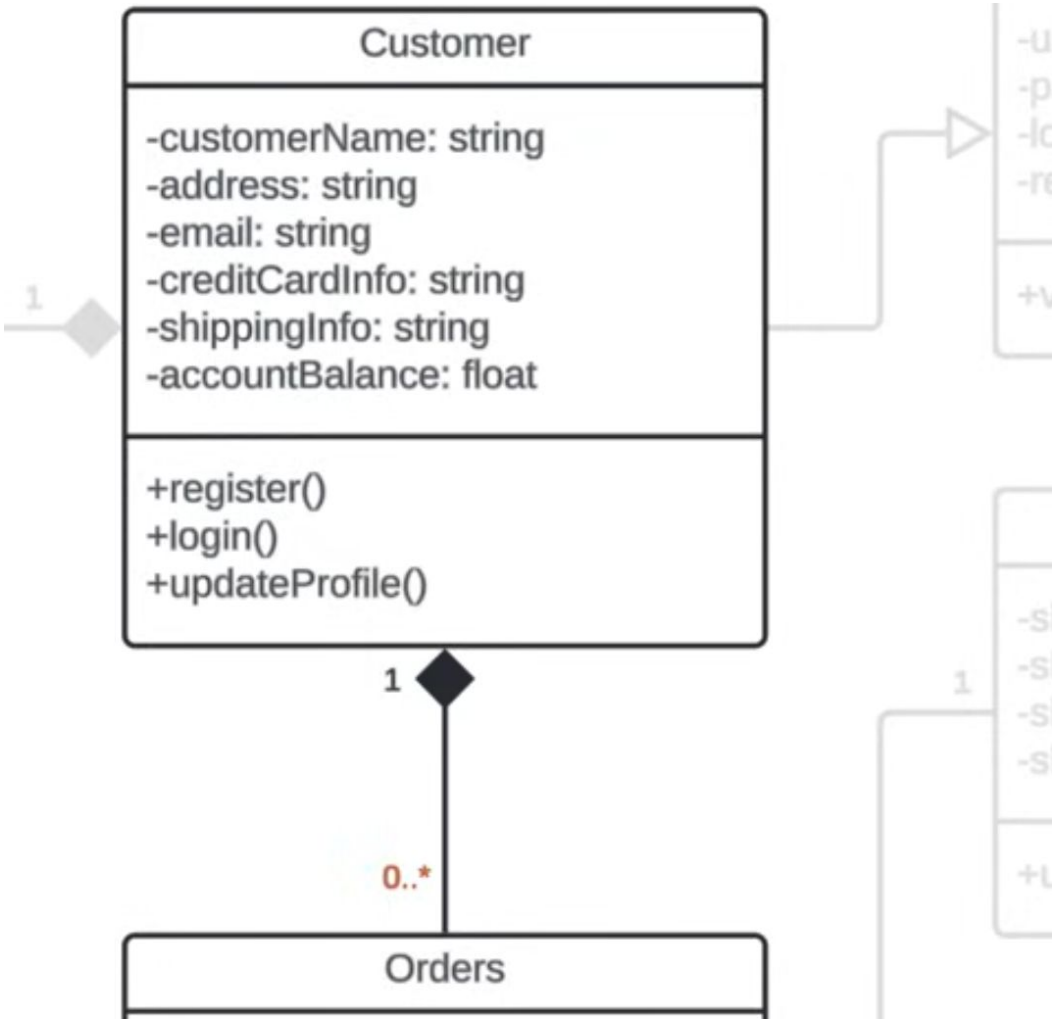
Composition: HAS-A

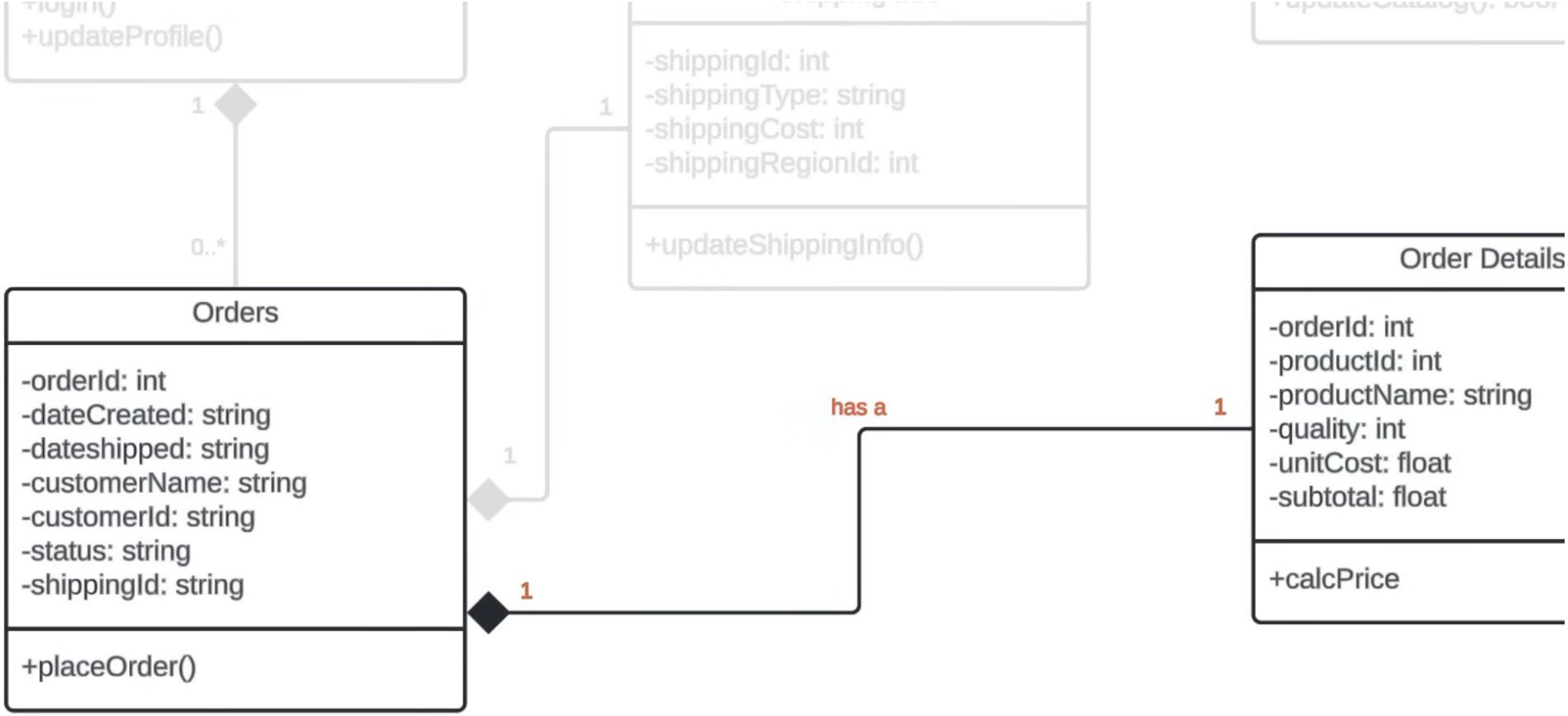

Customer
-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
1
0..*
Shopping Cart
-cartId: int
-productId: int
-quantity: int
-dateAdded: int
+addCartItem()
+updateQuantity()
+viewCartDetails()
+checkOut()
1
0..*
Orders
-orderId: int
-dateCreated: string
-dateshipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string
+placeOrder()
User
-userId: string
-password: string
-loginStatus: string
-registerDate: date
+verifyLogin(): bool
Administrator
-adminName: string
-email: string
+updateCatalog(): bool
Shipping Info
-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int
+updateShippingInfo()
Order Details
-orderId: int
-productId: int
-productName: string
-quality: int
-unitCost: float
-subtotal: float
+calcPrice
has a


Customer
-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
1
0..*
Shopping Cart
-cartId: int
-productId: int
-quantity: int
-dateAdded: int
+addCartItem()
+updateQuantity()
+viewCartDetails()
+checkOut()
1
0..*
Orders
-orderId: int
-dateCreated: string
-dateshipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string
+placeOrder()
Composition
When a child object wouldn't be able to exist without its parent object.


-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
+verifyLogin(): bool
Administrator
-adminName: string
-email: string
+updateCatalog(): bool
Shipping Info
-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int
+updateShippingInfo()
1
0..*
1
1
Orders
-orderId: int
-dateCreated: string
-dateshipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string
+placeOrder()
has a
1
1
Order Details
-orderId: int
-productId: int
-productName: string
-quality: int
-unitCost: float
-subtotal: float
+calcPrice

Multiplicity


Customer
-customerName: string
-address: string
-email: string
-creditCardInfo: string
-shippingInfo: string
-accountBalance: float
+register()
+login()
+updateProfile()
1
1
0..*
Orders


+updateProfile()
1
0..*
1
-shippingId: int
-shippingType: string
-shippingCost: int
-shippingRegionId: int
+updateShippingInfo()
Orders
-orderId: int
-dateCreated: string
-dateshipped: string
-customerName: string
-customerId: string
-status: string
-shippingId: string
+placeOrder()
1
1
has a
1
Order Details
-orderId: int
-productId: int
-productName: string
-quality: int
-unitCost: float
-subtotal: float
+calcPrice

What we have learned

What UML class diagrams look like

How to represent a class together with its attributes and methods

How to display relationships among different classes

Two types of relationships learned

- Inheritance (IS-A)
- Composition (HAS-A)

Denoting multiplicity in composition relationships

Let’s do some exercises

https://lucid.app/users/login

Lucid

Log in to access the Lucid Visual Colla

Email or username

Remember me

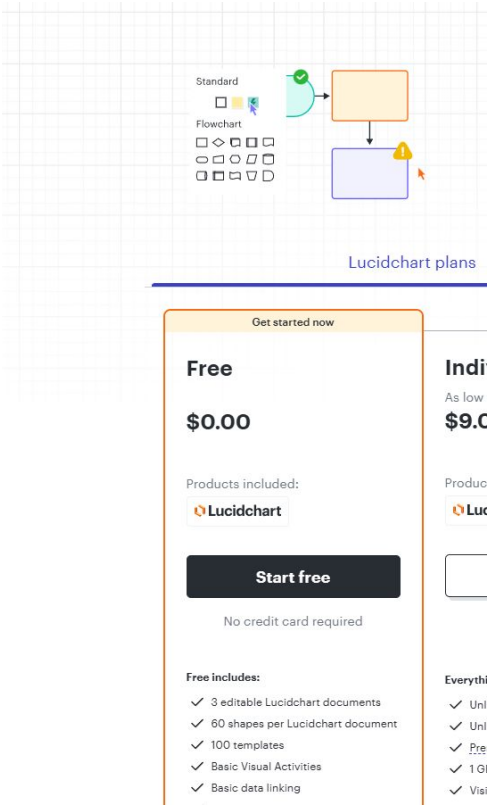
Log in with password

Next

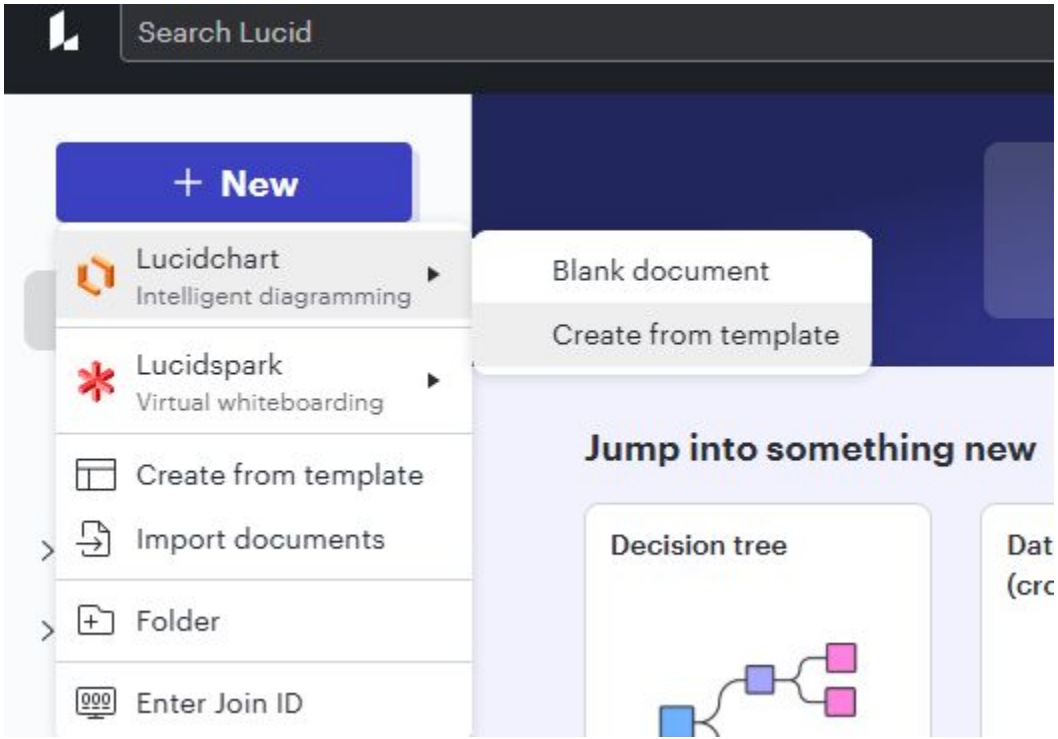
or

Log in with Google

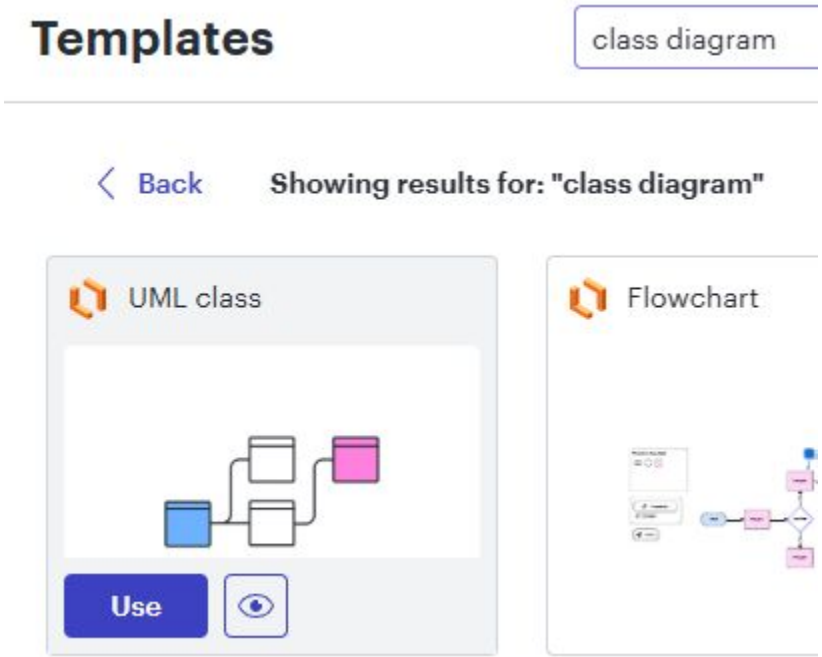
Select “Start free”



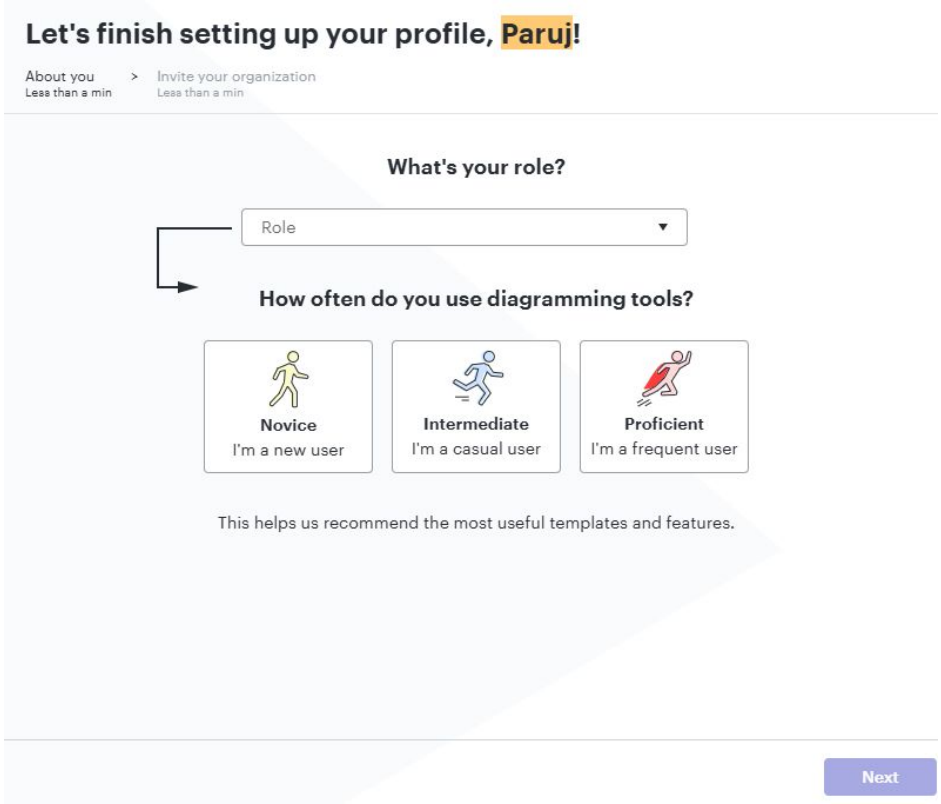
Click New->Lucidchart->Create from template



Search for “class diagram” and select UML class

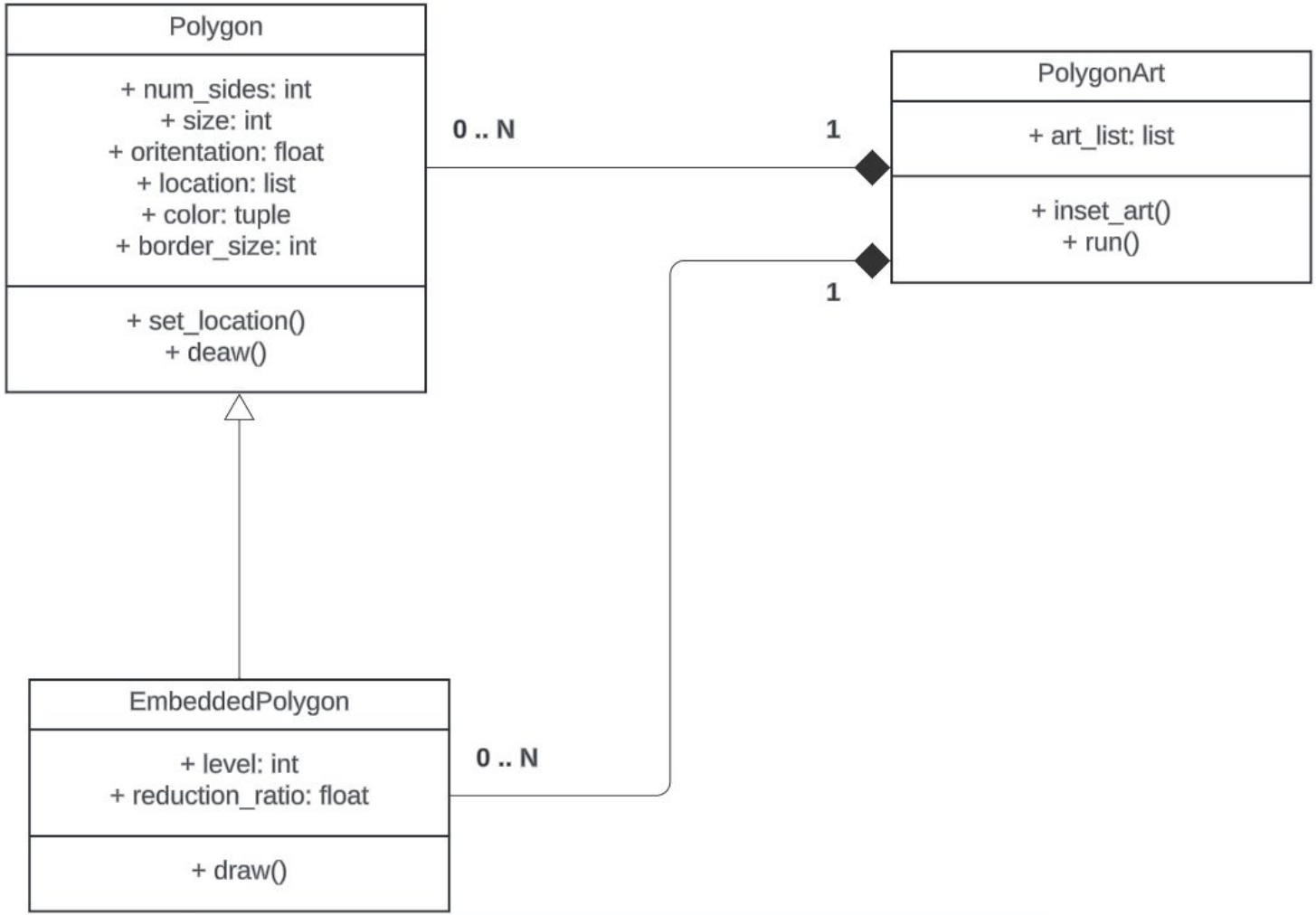


You may have to set up your profile first



Let’s create a UML class diagram for our OO programs with Turtle graphics

turtle_code


Polygon
+ num_sides: int
+ size: int
+ oritentation: float
+ location: list
+ color: tuple
+ border_size: int
+ set_location()
+ deaw()
0 .. N
1
PolygonArt
+ art_list: list
+ inset_art()
+ run()
1
EmbeddedPolygon
+ level: int
+ reduction_ratio: float
+ draw()
0 .. N

Let’s create a UML class diagram for the following code

```python
from datetime import datetime

class Person:
    def __init__(self, name, email):
        self._name = name
        self._email = email

    def get_contact_info(self):
        return f"{self._name} <{self._email}>"

class Student(Person):
    def __init__(self, name, email, student_id):
        super().__init__(name, email)
        self._student_id = student_id
        self._enrollments = []  # list of Enrollment objects

    def enroll(self, course):
        enrollment = Enrollment(self, course)
        self._enrollments.append(enrollment)
        return enrollment

    def list_courses(self):
        return [e.course.title for e in self._enrollments]

class Instructor(Person):
    def __init__(self, name, email, office):
        super().__init__(name, email)
        self._office = office
        self._courses = []   # teaches multiple Course objects

    def assign_course(self, course):
        self._courses.append(course)
```

```python
class Course:
    def __init__(self, code, title, instructor=None):
        self.code = code
        self.title = title
        self.instructor = instructor  # Instructor object
        self._enrollments = [] # list of Enrollment objects

    def add_enrollment(self, enrollment):
        self._enrollments.append(enrollment)

class Enrollment:
    def __init__(self, student, course):
        self.student = student      # Student object
        self.course = course        # Course object
        self.date = datetime.now()
        course.add_enrollment(self)
```