Class Diagrams

When you work on a project, you might need a proper visual representation of a system structure.

If you can divide your project into a set of objects and use the Object-Oriented Paradigm (OOP), you can use a **Class Diagram**.

It would help you to describe the components of your application and better understand the connections between them.

If you are working on an application that uses OOP, a Class Diagram will define your project structure and will make it apparent to the developers.

What is a class diagram?

In the Unified Model Language (UML), **Class Diagram** is a visual representation of an object-oriented structure. To describe said structure the diagram uses the following elements:

- Classes;
- Class attributes;
- Class methods;
- Class relationships.

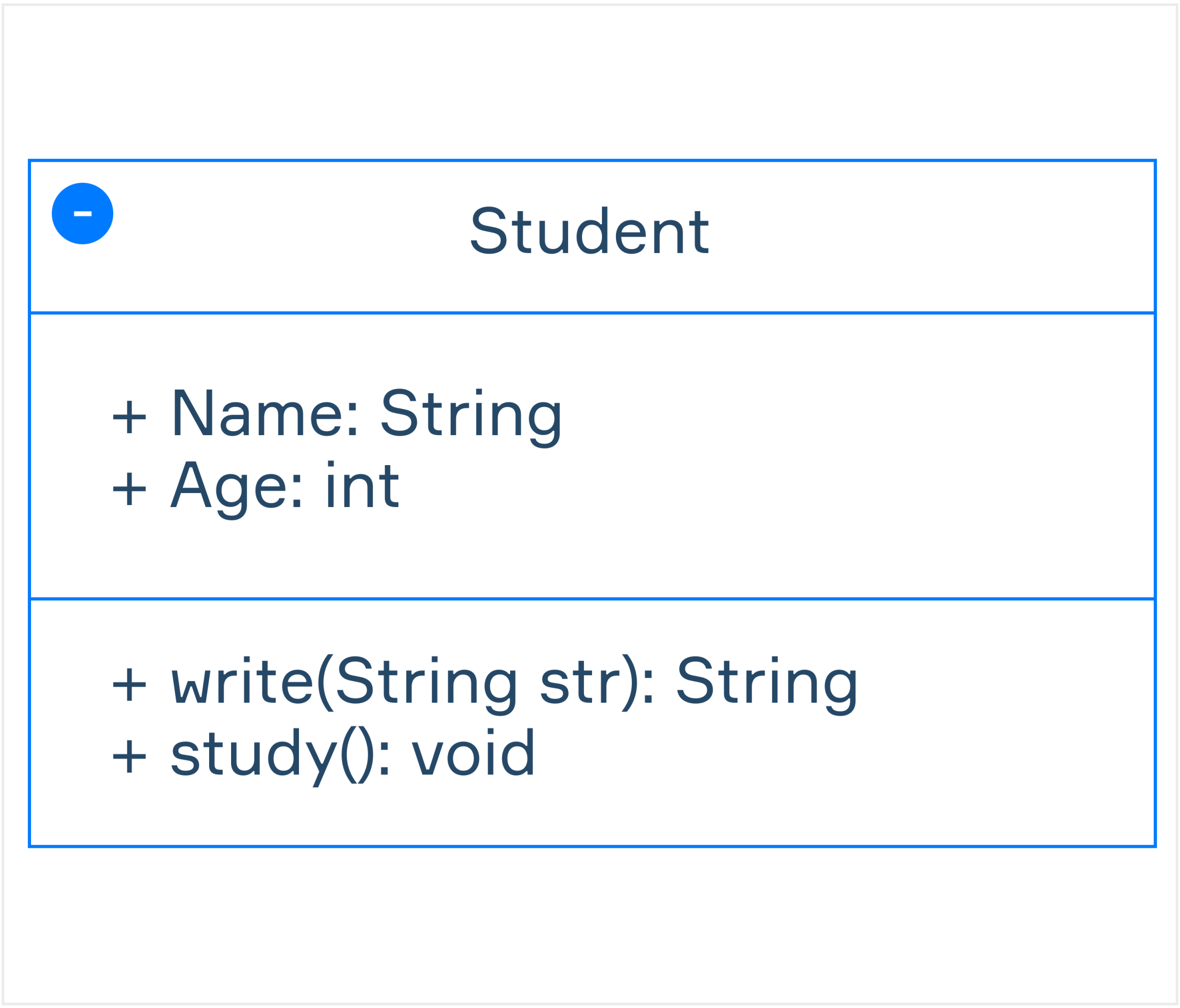
These elements form a set of classes and relationships between them.

Such structure provides a better understanding of connections within a system and allows us to easily demonstrate the contents of an object.

Description of a class

A class is a representation of an object that describes its methods and attributes. Simply put, a class can be seen as a blueprint for an object.

When the system creates a specimen of an object, it will be created according to a template that was described by the class.



Student has 2 attributes: *Name* defined by String type, and *Age* defined by int type. The class also has 2 methods: *write()* which returns String type and *study()* which returns nothing. Methods also have brackets that can be filled in with some parameters. In this example, we have the parameter *String str* that is required to activate method write*()*.

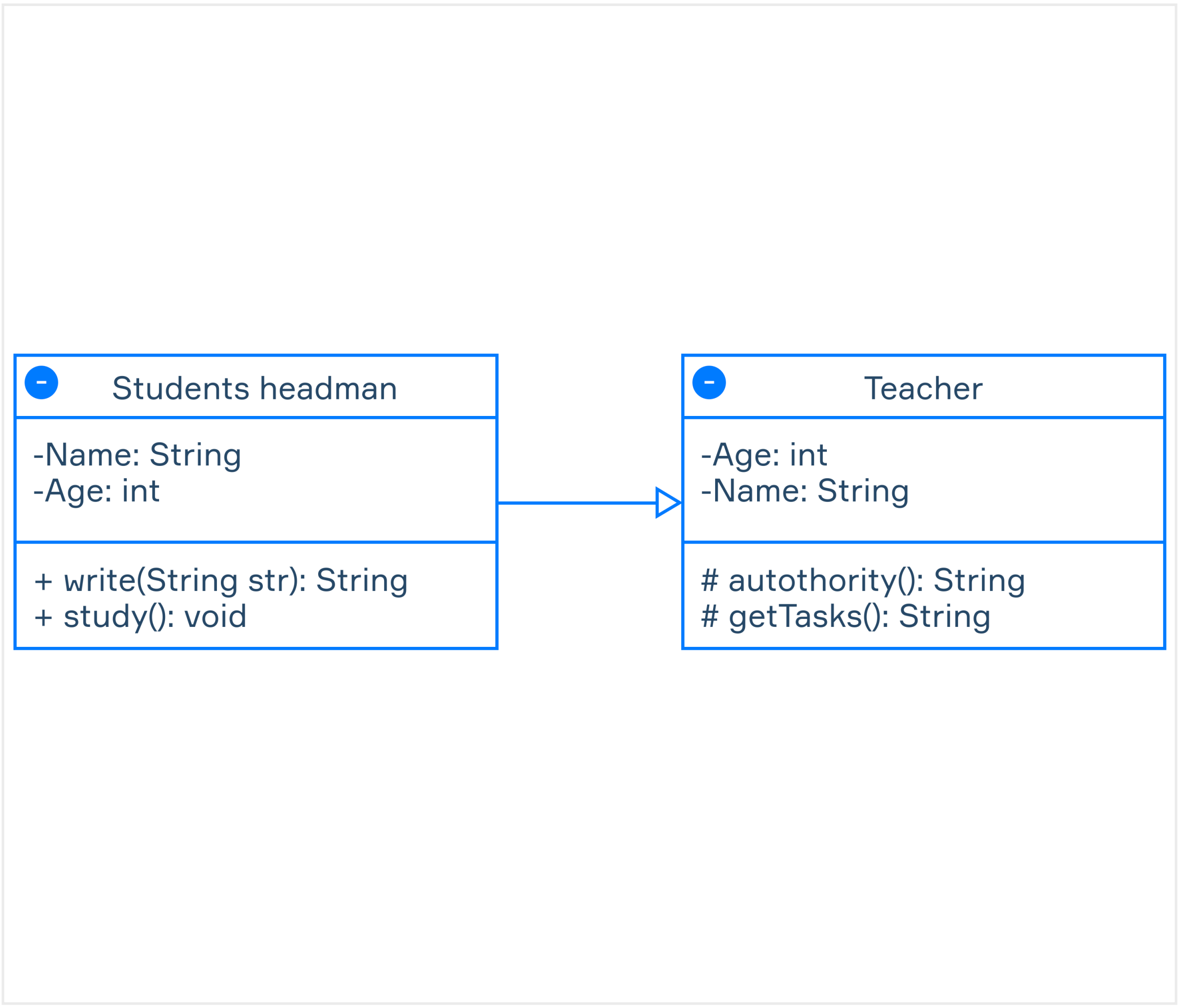
Each element of a class has a visibility option, defined before the name of the element. In this example, all the elements have a "+" before their names. That means that all the visibility options of these elements are set to **public**. Let's elaborate on these visibility options:

- + (**Public**) – element can be accessed by any class in the system;
- - (**Private**) – element can be accessed only by a class that owns it;
- # (**Protected**) – element can be accessed by classes that have a generalization (or inheritance) relationship with its class;
- ~ (**Package**) – element can be accessed by classes that are located in the same package.

Description of class relationships

Class relationships are a concept that defines connections between classes.

For example, we have 2 classes in our applications: *Student headman* and *Teacher*.

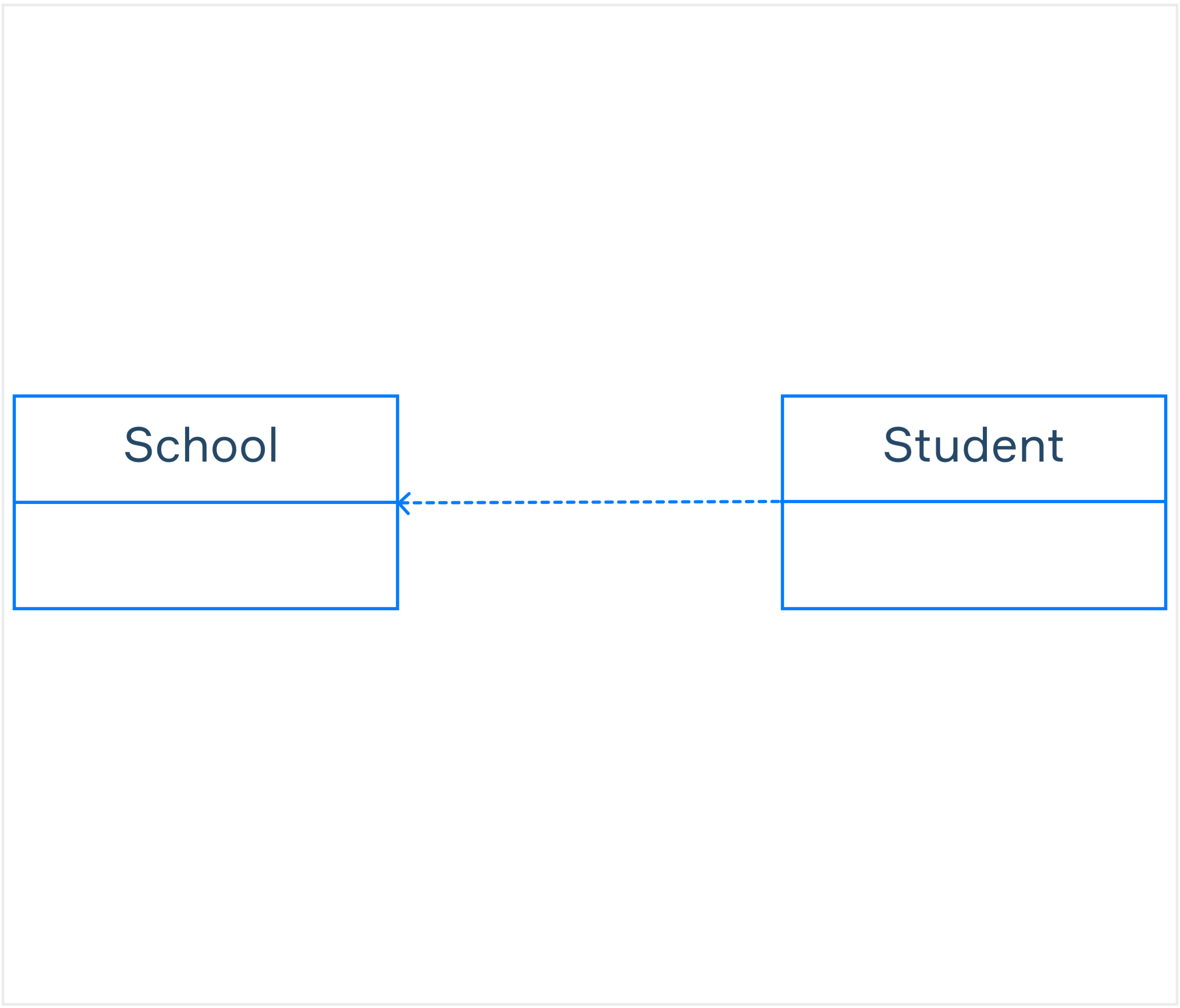


we use the type of relationship called **generalisation (or inheritance)**.

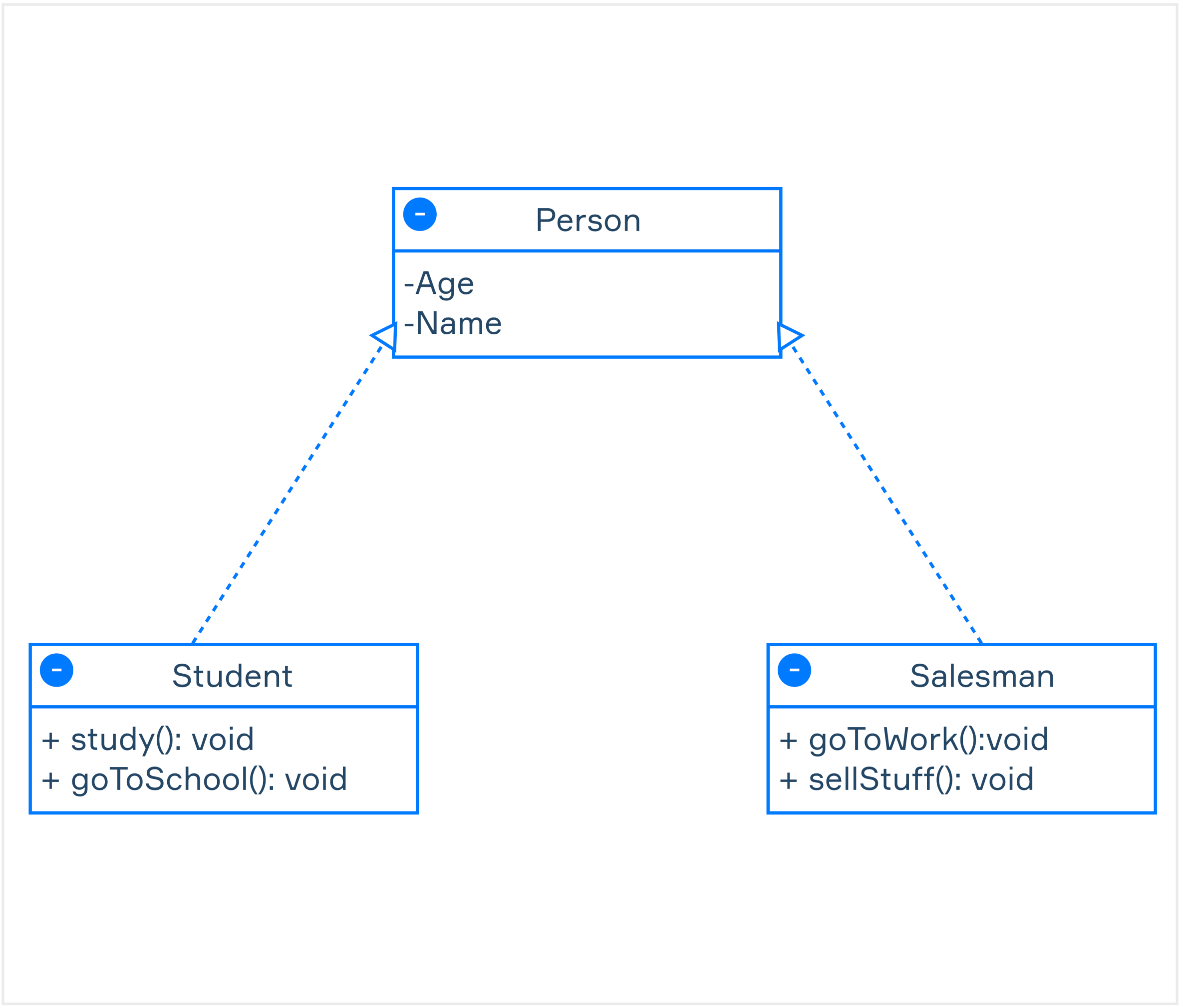
Let's elaborate on the relationship types:

- **Generalization (inheritance) –** a type of relationship where one class could be described as a child class which assumes and could use methods of a parent class. In our case *Student headman* is the child class of *Teacher.* This relationship could be visualized by this arrow below:

- **Dependency –** a type of relationship that indicates that some change in one class can affect another class. Here's an example of *Student* depending on *School*:

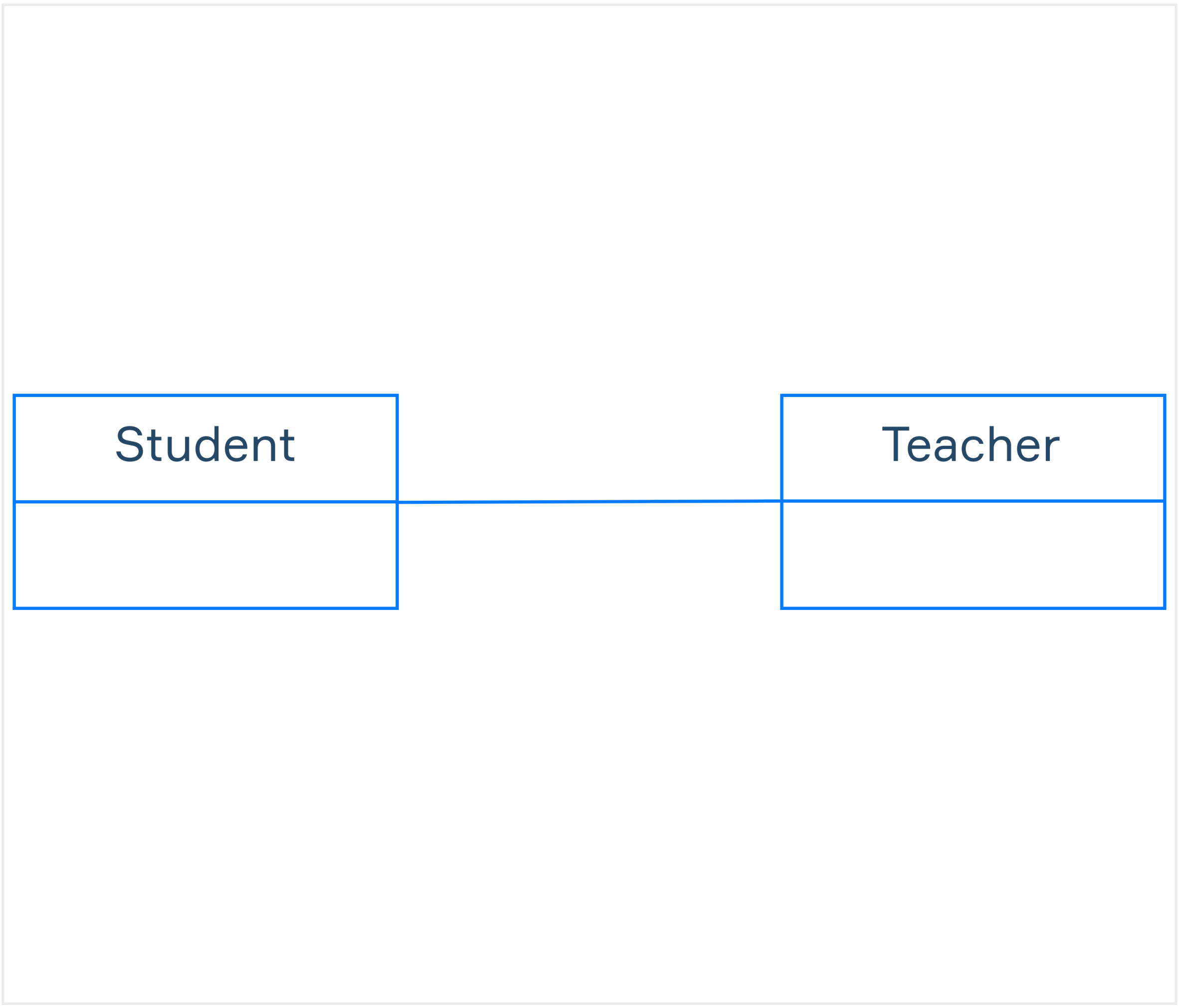


Realization – the relationship between the blueprint class and the object containing its respective implementation level details. For example, we have a class *Person* which describes the basic attributes of a person. It's a blueprint that can be made into an object that represents a specific person like a *Student*, or a *Salesman*. Our *Student* class can study and go to school, and our *Salesman* class can go to work and sell stuff, but they are people, so they will have *Age* and *Name* attributes.

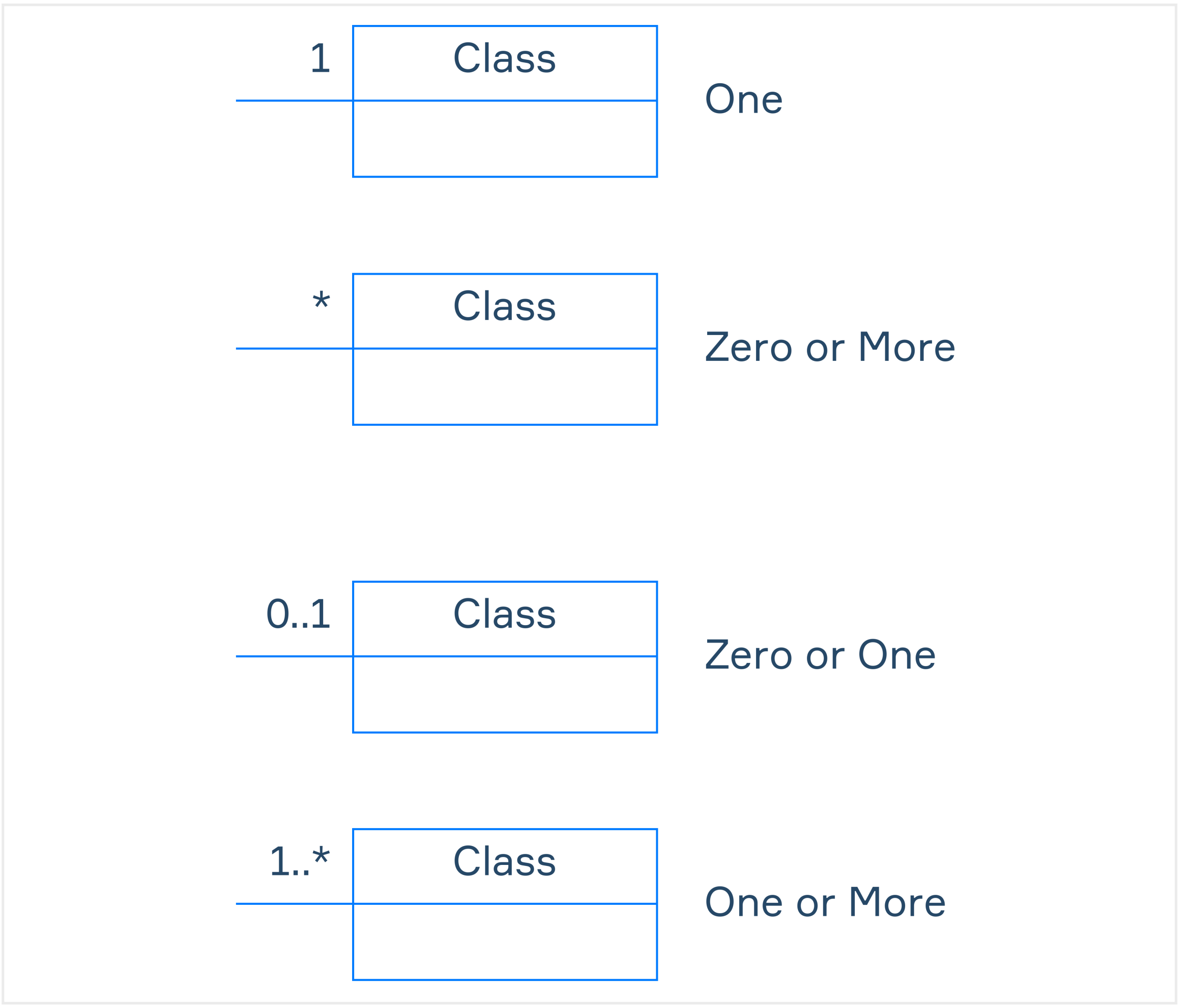


Description of association relationships

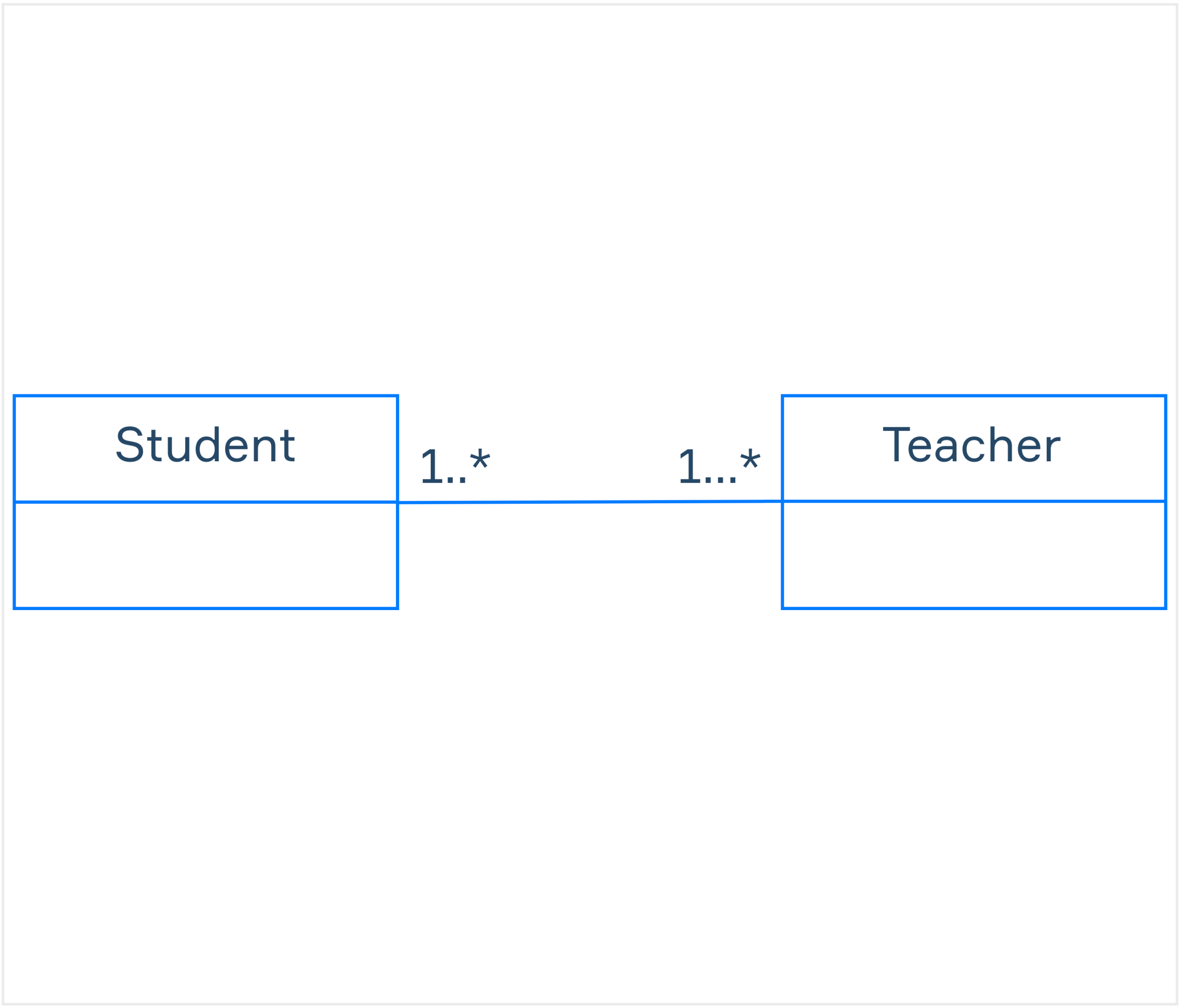
Association – a type of relationship that indicates that instances of one class are connected to instances of another. For example, *Teacher* teaches *Student*. This relationship can be visualised by a straight line:



Association relationships could also include a **cardinality** attribute. Simply put, this attribute defines the number of instances of a class that could exist in this relationship.

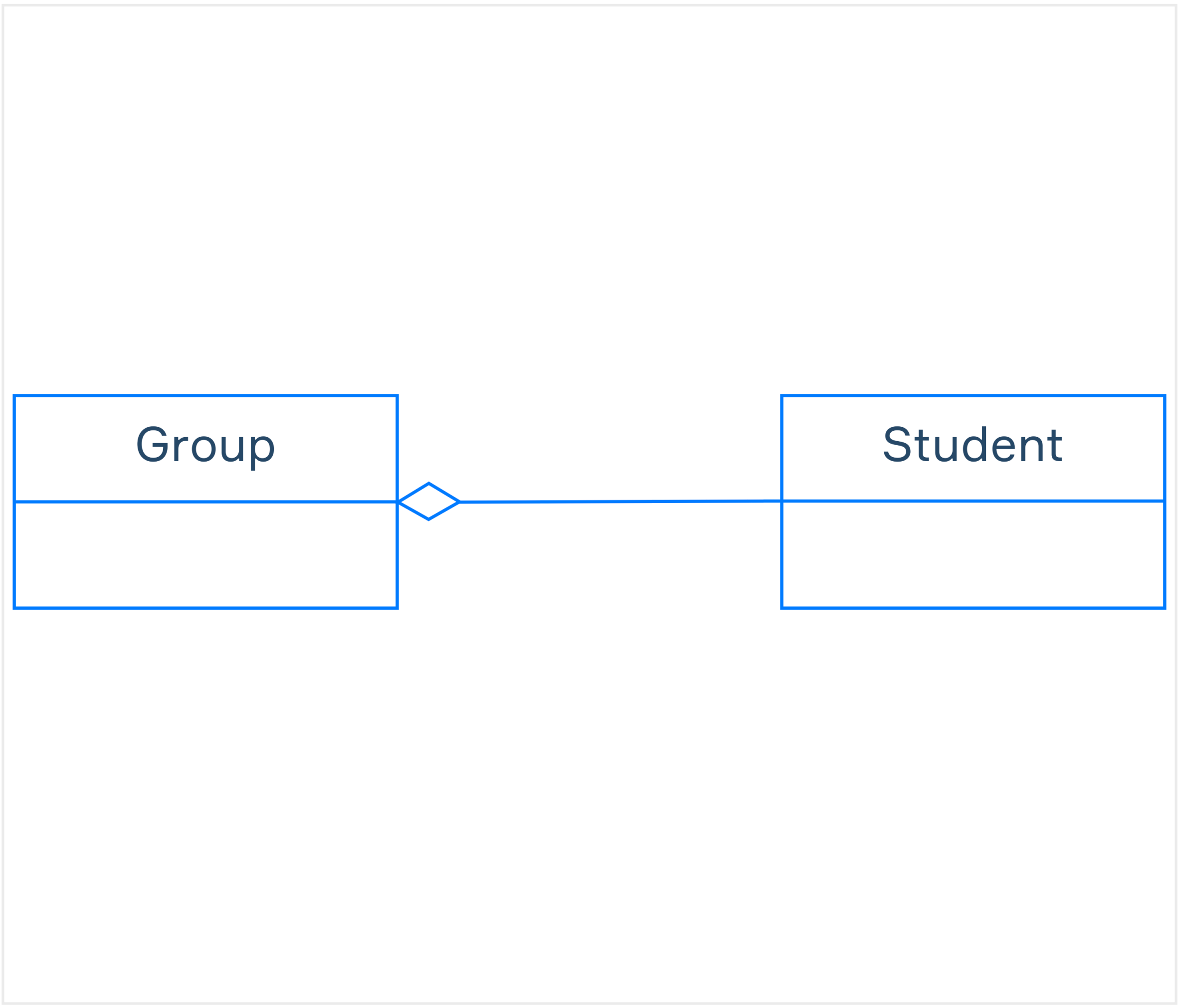


A basic example of this concept is this: class *Teacher* and class *Student* are connected by an association relationship. It means that *Teacher* teaches *Student*. One student can have one or multiple teachers, and the same can be said about teachers. So the diagram that describes this relationship would look like this:

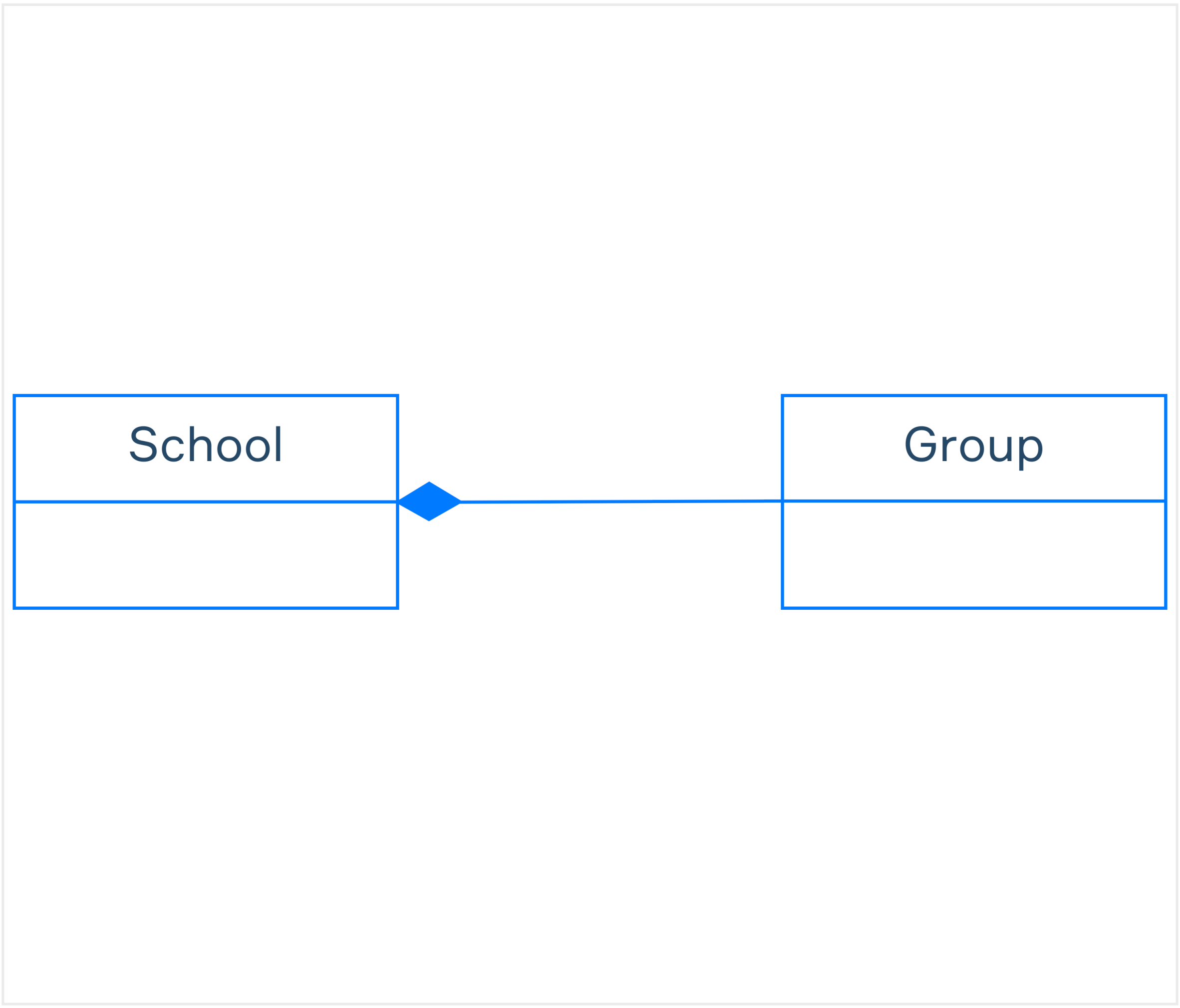


Association relationships also have two special types:

- **Aggregation** – a special type of association relation that describes one class as a part of the other. Classes in this relation have a separate lifespan. If we come back to our *Student* example, we can describe class *Student* as part of *a Group*. This relation can be represented in this way:



- **Composition –** a special type of aggregation, where classes share lifespan. If the main class (*School*) stops its functions, the class that is a part of it (*Group*) will stop functioning too. This relationship can be visualized like this:


School
Group