UML

The **Unified** (or **Universal) Modeling Language** (UML) is used by analysts and developers for modeling object-oriented applications. All phases of the Systems Development Life Cycle can be represented using UML diagramming techniques.

The objective of UML is to provide a common vocabulary of object-based terms and diagramming techniques that are rich enough to model any systems development project from analysis through implementation. There are many different types of UML diagrams. Here are a few examples:

UML Diagram	Shows	Used For
Use Case	Interaction between external users and the system	capture overall business requirements
Class	Nature of the system	To show the objects and the relationships
Sequence	Interaction between classes	To model the behavior
Component	Physical components such as dll files	To show the structure of the software
Deployment	Overall structure of a software system	Maps the software/hardware

UML Class Diagram relates to the code of the class structure

This diagram helps us to specify, visualize, construct, and document the various classes of a software system. A class diagram can have up to three sections that specify the following information about a class:

1. Class name
   a. *Abstract* class names are italicized
   b. **Sealed** class names are bolded
2. Fields and Properties
3. Methods

The entries in the bottom two sections contain the following details:

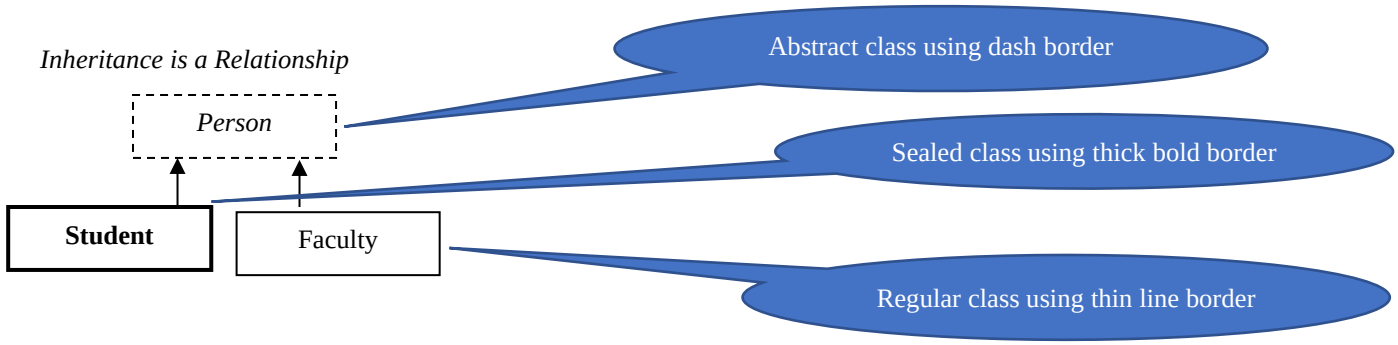
1. Accessibility:
   a. **+ plus** symbol for public access
   b. **– minus** symbol for private access
   c. **# pound** symbol for protected access
   d. **~ tilde** symbol for internal access
2. The member name (and parameter list for a method)
3. The data type of the member (or return type for a method)

Note: Use a **colon** to separate the member names and types; <u>underline</u> members that are *const* or *static*.

Class Diagram Template:

Class Name
Properties and Fields
Methods

UML Example



Code based on UML Class Diagram:

```
// Base Class: Person
public abstract class Person : Object  // abstract -> Generalization - use only as base class
{
    // private instance data
    // private - only accessable through public properties
    // instance - each object will have their own copy

    private string name = "";
    private DateTime birthdate;

    // no parameter constructor
    public Person() : base()
    {
        this.FullName = "Unknown";
        this.birthdate = DateTime.Today;

    }

    // three parameter constructor
    public Person(string fullName, DateTime birthDate) : base()
    {
        this.FullName = fullName;
        this.birthdate = birthDate;
    }

    // properties - used to access private class variables
    public string FullName
    {
        get
        {
            return this.name;
        }
        set
        {
            //empty string is not allowed
            if (value.Length > 0)
                this.name = value;
            else
                this.name = "Unknown";
        }
    }

    // readonly properties - no 'set' block
    public string BirthDate
    {
        get
        {
            return this.birthdate.ToLongDateString();
        }
    }

    // override Object class's ToString() method
    public override string  ToString()
    {
        return "Name: " + this.FullName + "\nBorn: " + this.BirthDate;
    }
}
```

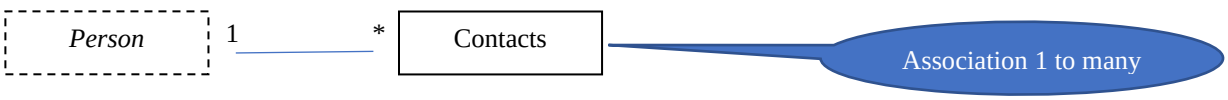
Person
- name : string - birthdate : DateTime + FullName : string + BirthDate : string
+ ToString() : string + Person() + Person(string, DateTime)

Relationships between classes:

In addition to Inheritance there are two other principal kinds of relationships:

Association - represent relationships between instances of types (one to many, many to many)

Aggregation - Aggregation, a form of object composition or containment in object-oriented design. The Person class above contains a DateTime type. A UWP Page contains controls such as Buttons and Textboxes.



Another Example:

```
//Derived Class Example: Student (Base Class - Person)
public sealed class Student : Person  // sealed -> can't use as a base class
{
    // instance variables
    private string studentNumber;
    private int weightedAverage;

    // class variable
    public const string SCHOOL = "Niagara College";

    // Constructors
    public Student()
        : base()
    {
        studentNumber = "0000000";
        weightedAverage = 0;
    }
    public Student(string fullName, DateTime birthDate, string studentNumber, int average)
        : base(fullName, birthDate)
    {
        this.studentNumber = studentNumber;
        weightedAverage = average;
    }
    // Properties
    public string StudentNumber
    {
        get
        {
            return studentNumber;
        }
        set
        {
            try
            {
                //make sure the student number is numeric
                Convert.ToInt32(value);
                //make sure the student number is 7 characters
                if (value.Length == 7)
                    studentNumber = value;
                else
                    studentNumber = value.PadRight(7, '0');
            }
            catch
            {
                studentNumber = "0000000";
            }
        }
    }
    public int WeightedAverage
    {
        get
        {
            return weightedAverage;
        }
        set
        {
            if (!(value < 0 || value > 100))
                weightedAverage = value;
        }
    }
    public override string ToString()
    {
        return base.ToString() + "\nStudent Number: " + StudentNumber +
               "\nWeighted Average: " + WeightedAverage;
    }
}
```

Student
- studentNumber : string - weightedAverage : int + SCHOOL : string + StudentNumber : string + WeightedAverage : int
+ ToString() : string + Student() + Student(string, DateTime, string, int)

More Information: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/