CLASS DIAGRAMS
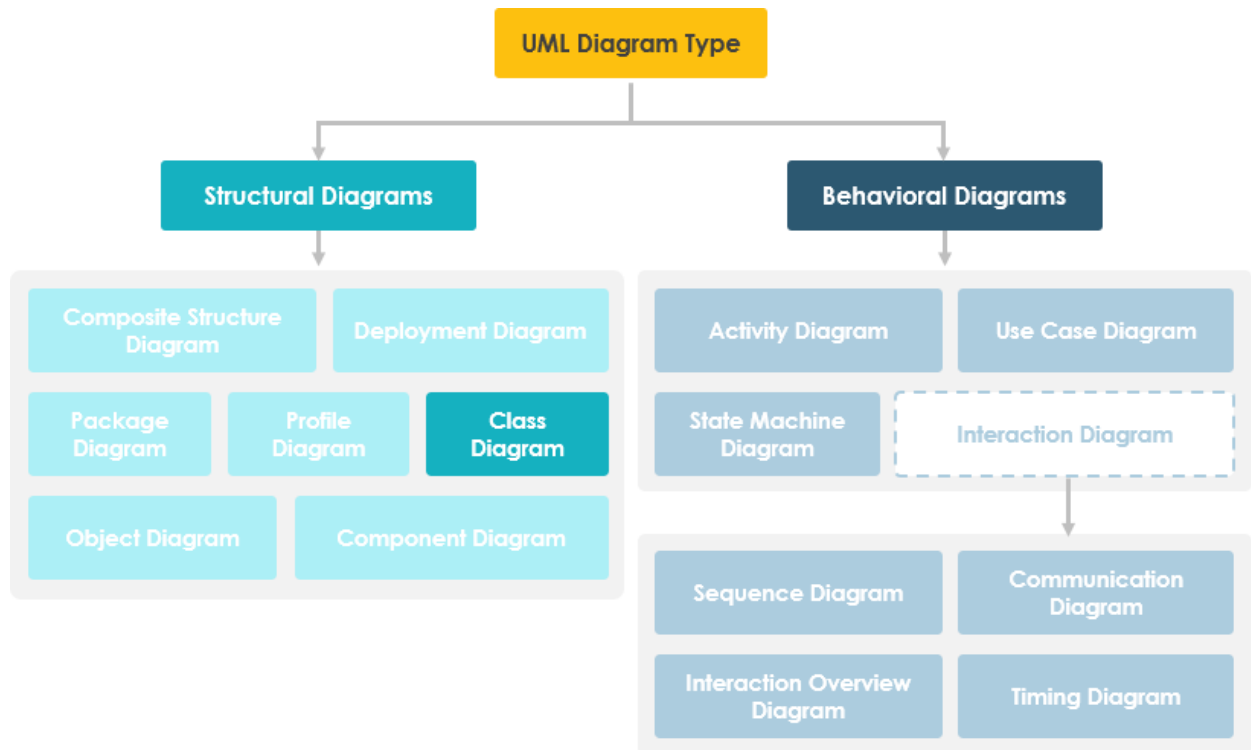
In software engineering, a class diagram in the [Unified Modeling Language (UML)](#) is **a type of static structure diagram** that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Purpose of Class Diagrams

1. Shows static structure of classifiers in a system
2. Diagram provides basic notation for other structure diagrams prescribed by UML
3. Helpful for developers and other team members too
4. Business Analysts can use class diagrams to model systems from business perspective

A UML class diagram is made up of:

- A set of classes and
- A set of relationships between classes

What is a Class

A description of a group of objects all with similar roles in the system, which consists of:

- **Structural features** (attributes) define what objects of the class "know"

- o Represent the state of an object of the class
- o Are descriptions of the structural or static features of a class
- **Behavioral features** (operations) define what objects of the class "can do"
  - o Define the way in which objects may interact
  - o Operations are descriptions of behavioral or dynamic features of a class

Class Notation

A class notation consists of three parts:

1. **Class Name**
   - o The name of the class appears in the first partition.
2. **Class Attributes**
   - o Attributes are shown in the second partition.
   - o The attribute type is shown after the colon.
   - o Attributes map onto member variables (data members) in code.
3. **Class Operations** (Methods)
   - o Operations are shown in the third partition. They are services the class provides.
   - o The return type of a method is shown after the colon at the end of the method signature.
   - o The return type of method parameters are shown after the colon following the parameter name.
   - o Operations map onto class methods in code

MyClass
+attribute1 : int
-attribute2 : float
#attribute3 : Circle
+op1(in p1 : bool, in p2) : String
-op2(input p3 : int) : float
#op3(out p6) : Class6*

The graphical representation of the class - MyClass as shown above:

- MyClass has 3 attributes and 3 operations
- Parameter p3 of op2 is of type int
- op2 returns a float
- op3 returns a pointer (denoted by a *) to Class6
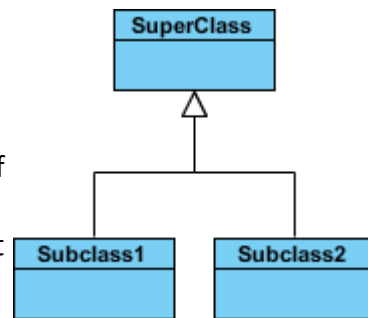
Class Relationships

A class may be involved in one or more relationships with other classes. A relationship can be one of the following types: (Refer to the figure on the right for the graphical representation of relationships).

Relationship Type   **Graphical Representation**

Inheritance (or Generalization):

- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializations of Super Class.
- A solid line with a hollow arrowhead that point from the child to the parent class
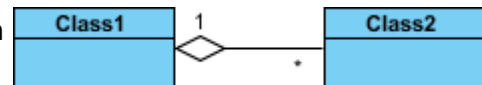

Simple Association:

- A structural link between two peer classes.
- There is an association between Class1 and Class2
- A solid line connecting two classes

Aggregation:

A special type of association. It represents a "part of" relationship.

- Class2 is part of Class1.
- Many instances (denoted by the *) of Class2 can be associated with Class1.
- Objects of Class1 and Class2 have separate lifetimes.
- A solid line with a unfilled diamond at the association end connected to the class of composite

Composition:

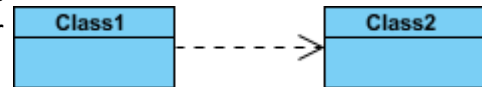A special type of aggregation where parts are destroyed when the whole is destroyed.

- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself.

- A solid line with a filled diamond at the association connected to the class of composite
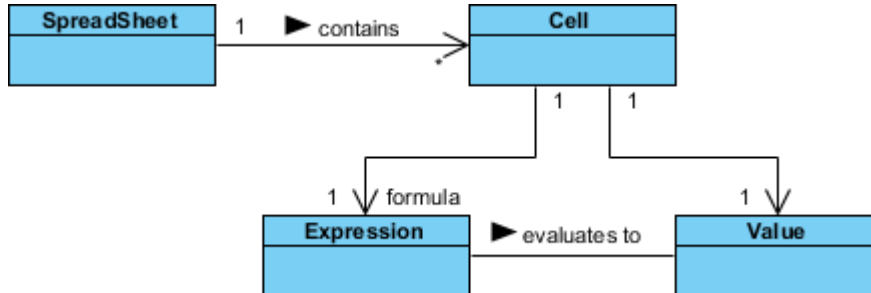
Dependency:

- Exists between two classes if changes to the definition of one may cause changes to the other (but not the other way around).
- Class1 depends on Class2
- A dashed line with an open arrow

Relationship Names

- Names of relationships are written in the middle of the association line.
- Good relation names make sense when you read them out loud:
  - "Every spreadsheet **contains** some number of cells",
  - "an expression **evaluates to** a value"
- They often have a **small arrowhead to show the direction** in which direction to read the relationship, e.g., expressions evaluate to values, but values do not evaluate to expressions.

Relationship - Roles

- A role is a directional purpose of an association.
- Roles are written at the ends of an association line and describe the purpose played by that class in the relationship.
  - E.g., A cell is related to an expression. The nature of the relationship is that the expression is the **formula** of the cell.

Navigability

The arrows indicate whether, given one instance participating in a relationship, it is possible to determine the instances of the other class that are related to it.

The diagram above suggests that,

- Given a spreadsheet, we can locate all of the cells that it contains, but that
    o we cannot determine from a cell in what spreadsheet it is contained.
- Given a cell, we can obtain the related expression and value, but
    o given a value (or expression) we cannot find the cell of which those are attributes.
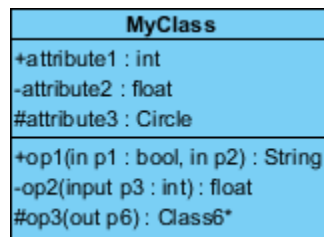
Visibility of Class attributes and Operations

In object-oriented design, there is a notation of visibility for attributes and operations. UML identifies four types of visibility: **public**, **protected**, **private**, and **package**.

The +, -, # and ~ symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations
- ~ denotes package attributes or operations

Class Visibility Example

In the example above:

☐ attribute1 and op1 of MyClassName are public
☐ attribute3 and op3 are protected.
☐ attribute2 and op2 are private.

Access for each of these visibility types is shown below for members of different classes.

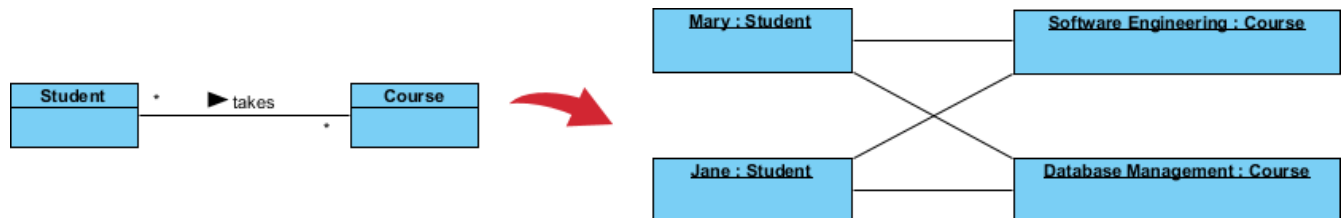| Access Right	public (+)	private (-)	protected (#)	Package (~)
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same package

Multiplicity

How many objects of each class take part in the relationships and multiplicity can be expressed as:

- Exactly one - 1
- Zero or one - 0..1
- Many - 0..* or *
- One or more - 1..*
- Exact Number - e.g. 3..4 or 6
- Or a complex relationship - e.g. 0..1, 3..4, 6.* would mean any number of objects other than 2 or 5
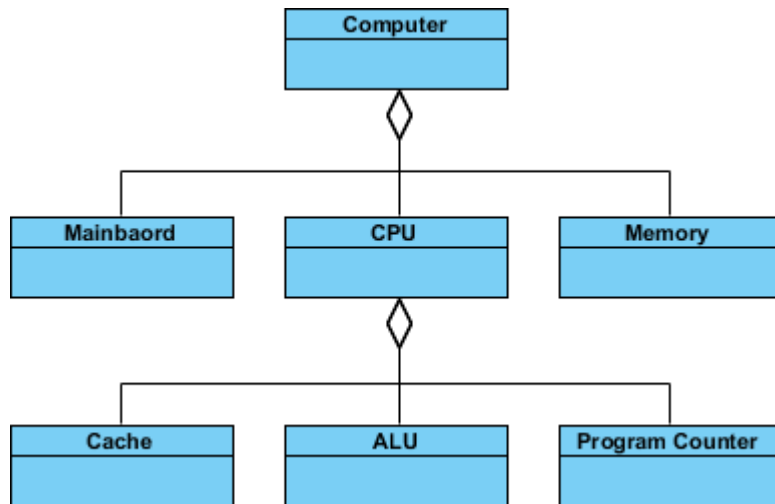
Multiplicity Example

- Requirement: A Student can take many Courses and many Students can be enrolled in one Course.
- In the example below, the **class diagram** (on the left), describes the statement of the requirement above for the static model while the object diagram (on the right) shows the snapshot (an instance of the class diagram) of the course enrollment for the courses Software Engineering and Database Management respectively)
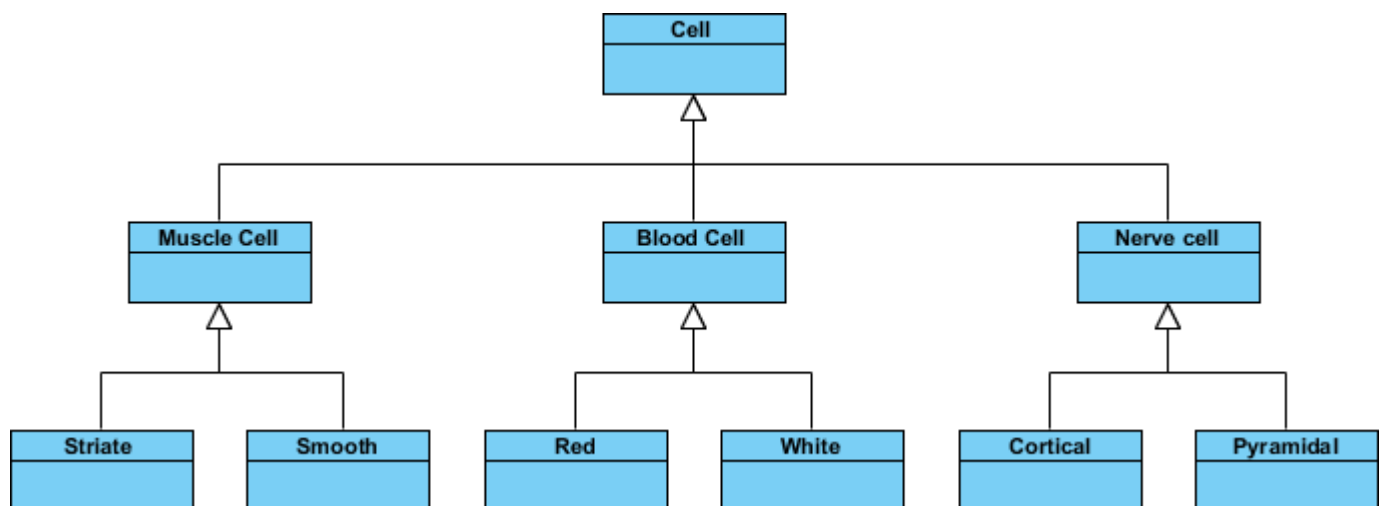
Aggregation Example - Computer and parts

- An aggregation is a special case of association denoting a "consists-of" hierarchy
- The aggregate is the parent class, the components are the children classes
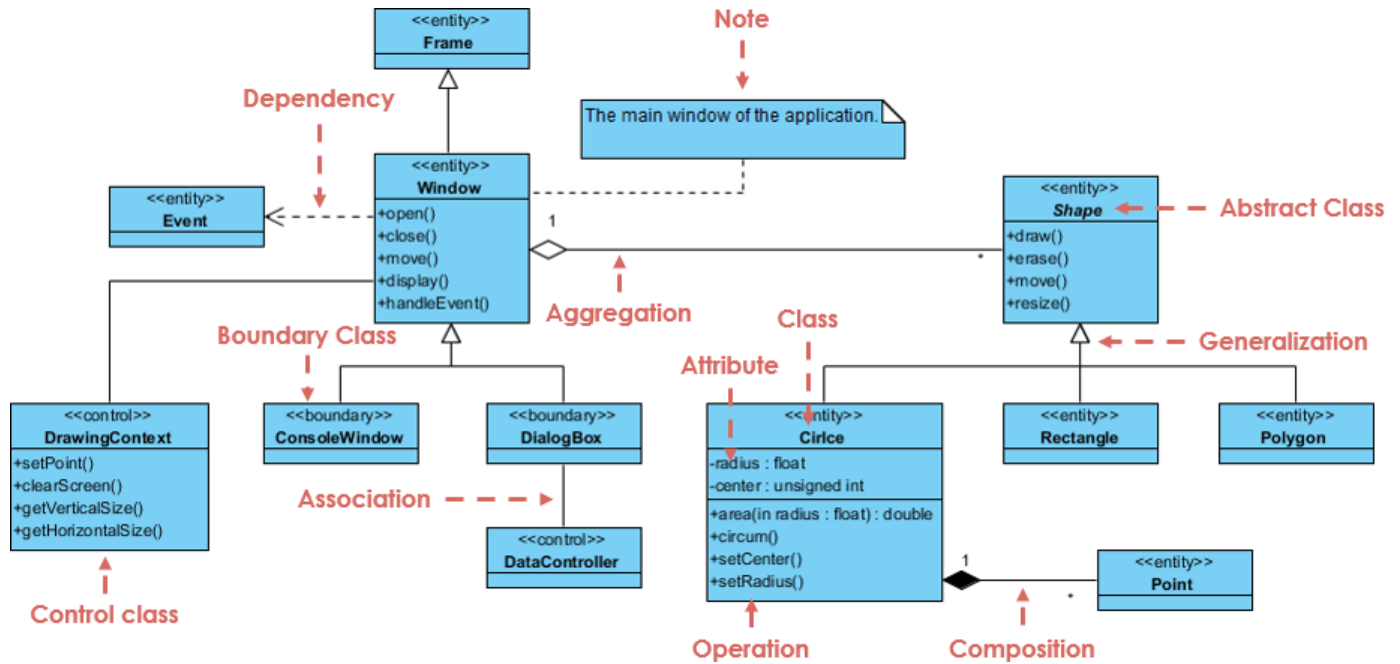

Inheritance Example - Cell Taxonomy

- Inheritance is another special case of an association denoting a "kind-of" hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The children classes inherit the attributes and operations of the parent class.

Class Diagram - Diagram Tool Example

A class diagram may also have notes attached to classes or relationships. Notes are shown in grey.

In the example above:

We can interpret the meaning of the above class diagram by reading through the points as following.

1. Shape is an abstract class. It is shown in Italics.
2. Shape is a superclass. Circle, Rectangle and Polygon are derived from Shape. In other words, a Circle is-a Shape. This is a generalization / inheritance relationship.
3. There is an association between DialogBox and DataController.
4. Shape is part-of Window. This is an aggregation relationship. Shape can exist without Window.
5. Point is part-of Circle. This is a composition relationship. Point cannot exist without a Circle.
6. Window is dependent on Event. However, Event is not dependent on Window.
7. The attributes of Circle are radius and center. This is an entity class.
8. The method names of Circle are area(), circum(), setCenter() and setRadius().
9. The parameter radius in Circle is an in parameter of type float.
10. The method area() of class Circle returns a value of type double.
11. The attributes and method names of Rectangle are hidden. Some other classes in the diagram also have their attributes and method names hidden.