Contents

A Beginners Guide to The Unified Modeling Language (UML)

What is the UML?

UML stands for "Unified Modeling Language". The UML is a modern diagrammatic language. It has become the standard way to design and document software. There are 13 different kinds of diagrams in the UML. In CSE201 we only talk about *class diagrams* and *object diagrams*. It is a required part of the Computer Science core at CSUSB.

Why use the UML? (1) Many people think in pictures. (2) You'll be able to share your ideas with others. (3) A picture can express a lot of words. (4) UML is becoming valuable in the job market. (5) You will think of more ideas and be able to spot bad ones.

How do I use the UML?

There are four ways people use the UML:

Table

Purpose	Tool
A rough sketch of an idea.	Chalk board. Pencil and paper. Hint. Leave boxes open to the right and bottom.
A blueprint of some software	A drawing program like *Dia* or *Visio*. Some word processors can also do graphics.
Generating a program	A CASE tool like Pegasus or Rational Rose
Required documentation	Either use a CASE tool to reverse engineer the diagram from code, or a drawing tool.

(Close Table)

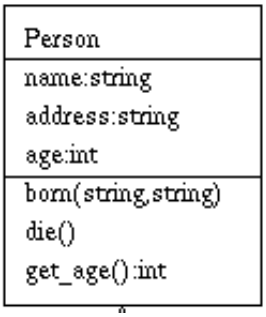
As a rule: **keep it simple.** Omit things on a diagram that are neither important nor interesting.

Only do a diagram if you can see some value for you or your client. For example it is worth planning the classes you will be programming and how they interact before you start coding them. You can often earn credit in CSE classes by drawing **UML** diagrams to explain what you have done. Many clients/companies have standards that require you to *document* your code to help other maintain it.

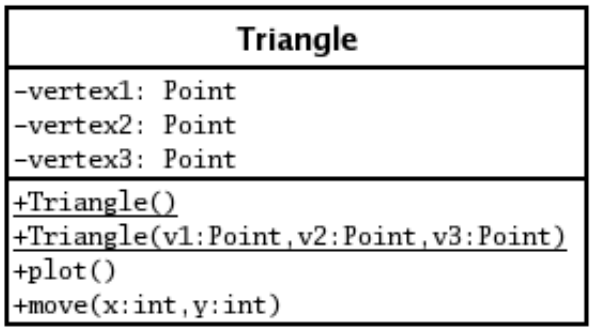
How do I draw a single class in the UML?

The UML can express most things you can code in C++. A single box represents a class of objects. The name of the class is at the top. In the next part of the box lists the data each object contains. In the third part of the class box list the operations (functions) that an object can carry out.

Here is a simple model of a person with a name, address, and age:



For example a triangle has three Points....

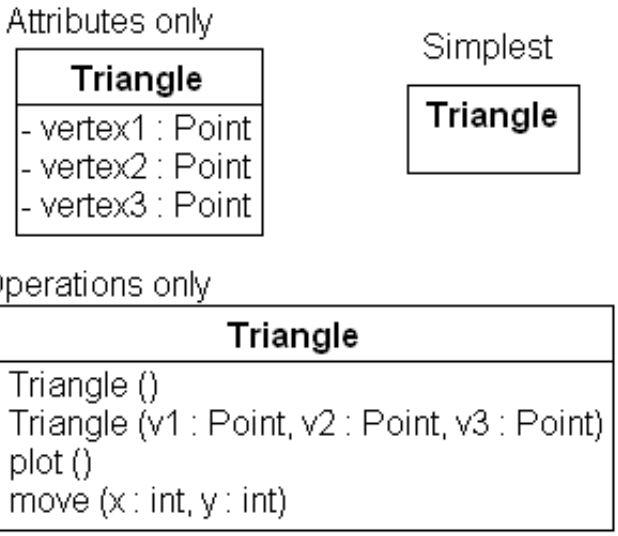


Notice that instead of the C++ 'string name;' we write 'name : string'. More, we omit the word *void*.

Notice that private items (like the vertexes above) are marked with a minus sign. Public items are marked with a plus sign.

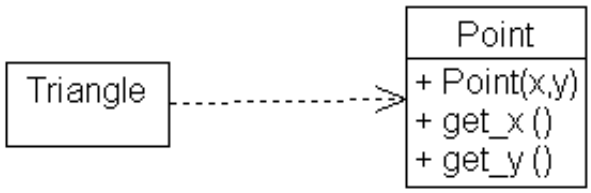
The constructors are underlined to show that they are not applied to objects. Some people omit the underline. You can also write <<*constructor*>> if you want to be very correct.

You don't have to show all the details unless you need to. You can omit the underlining of constructors. You can even omit constructors unless they are special in some way. Any of the two lower compartments can be left out. For example the following are also pictures of the Triangle class with details suppressed.

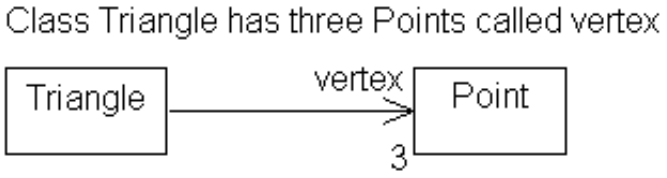


How do I show relations between my classes using UML?

Many classes can be put on one diagram and connected together. The following diagram says that we had to use the class Point to define a triangle:



We can be more precise and say that a Triangle must have three Points called vertexes:

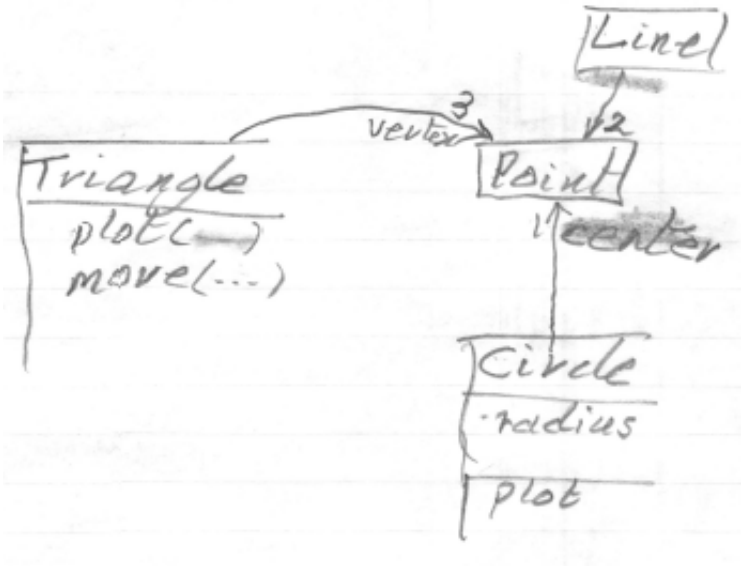


Can you show us an example of the UML that we can use in this class?

The diagram above is from a laboratory. You will be shown how to use *Dia* in our labs.

How do you start drawing diagrams?

Start with rough sketches. Omit more and use pencil or chalk.



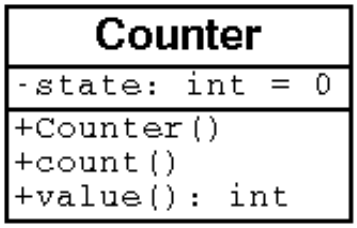
You can sketch a class in the UML quicker than you can write a C++ class.

How do I draw a single object in the UML?

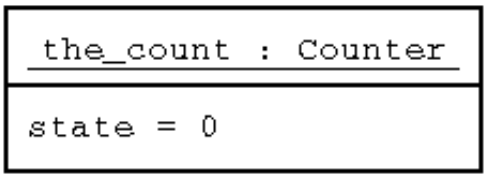
It helps to be able to draw objects as well as classes. These are called **object diagrams** of course.

The rules are simple: (1) use a box, (2) underline the name of the object and its class -- name:class, and (3) put attributes in a compartment under the name. Here for example is a simple Counter class plus object diagrams of of an object *the_count* (perhaps from Sesame St.?) before and after the *count* operation is applied to it.

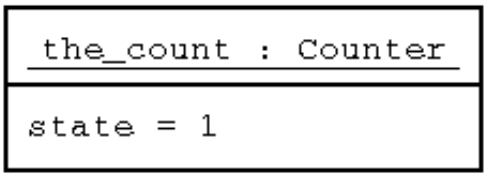
Suppose we've defined a class called Counter:



Then a recently constructed Counter object called the_count.



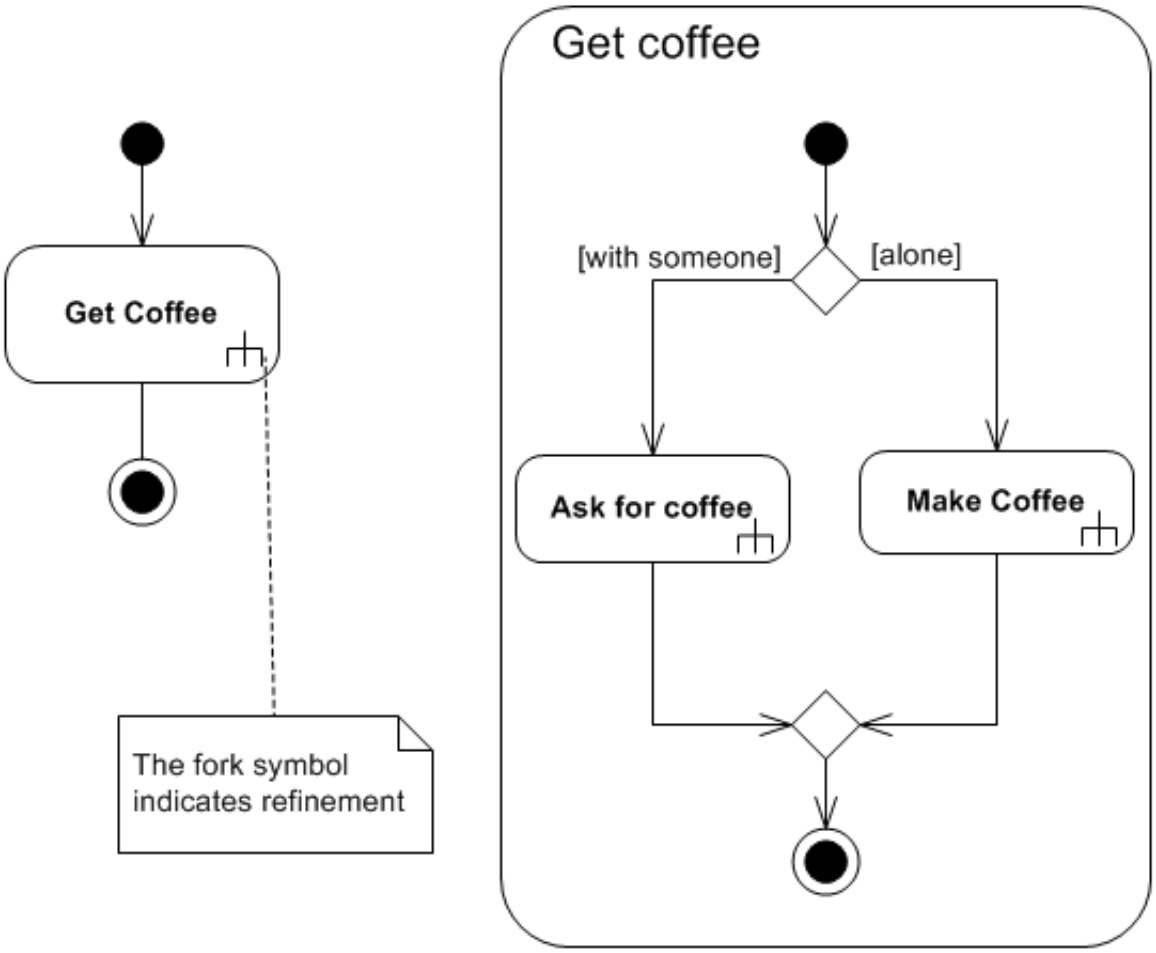
The same object after being sent a count() message



Some example code is in [**counter.cpp**] for example

How do I draw a complex algorithm in the UML?

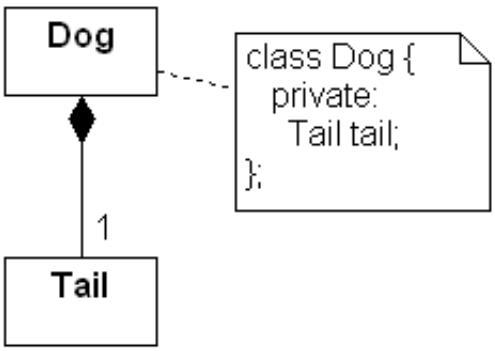
To plan a complex algorithm use an **Activity diagram** (like a traditional flowchart).



Notice -- the rounded corners on the activities, the diamonds where decisions are made or resolve, and the use of square brackets about conditions -- [*condition*].

I've seen a diagram with a funny diamond shape on it, what does it mean?

This is used to relate a whole to it's parts. It shows ownership. It is used when you have a "Has a" relationship like: "A dog has a tail".



This notation can be used to show that a class has a data member or field. It is called **composition.**

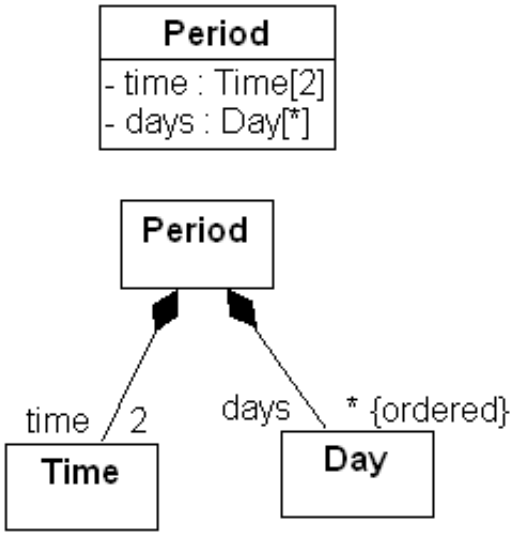
Can I show arrays and vectors in the UML?

Yes. You can show an array as an attribute with the size in brackets (just like in C++). Vectors can vary in size so you put an asterisk in the between the braces. You can also put these **multiplicities** on links between classes.

For an example, here is the code for a class called Period as part of an application modelling this campus:

```
class Period {
        private:
                Time time[2];//start and finish Times
                vector<Day> days;//Days in the week
};//end class Period
```

Here are a couple of valid diagrams that show that each Period contains a pair of times and variable number of days.



A simple composition with a multiplicity of one indicates a simple data member in the C++ class. A fixed multiplicity suggests an array. A range of multiplicities or an asterisk suggests a more complex data structure such as a vector, list, or set. If the order is important, then the constraint "{ordered}" should be added.

So the UML is all about drawing classes

NO. There are 13 different diagrams, each fine tuned to a particular purpose. This was a very quick introduction suitable for CSE201. We have mentioned class diagrams, object diagrams, and

activity diagrams. There is a lot more to learn about the UML. See [**Unified_Modeling_Language**] on the Wikipedia for a summary.

Where can I learn more about the UML?

Take CSE202 [**../cs202/uml0.html**] , CSE330, CSE375, and CSE455!

Here is a link to my local documentation on the UML: [**uml.html**]

In your future career you may need a text book. If so try: Martin Fowler and K. Scott, "UML Distilled: A Brief Guide to the Standard Object Modeling Language," 3rd Ed., Addison-Wesley, 2005.

Experiments on the UML

1. Use a pencil and paper sketch to think about your next project that has a class in it. Keep it rough! What have you learned about the project?
2. Use a rough pencil-and-paper sketch of an activity diagram of your next project that has a complex piece of logic or algorithm in it. What have you learned about the project?
3. Take any C++ program you have written that defines a class -- print out the code and draw a rough diagram of the class on the blank space. What did you learn?
4. Take a rough diagram and try producing a high quality version by using *Dia*, *Visio*, or any other drawing tool you have access to.

. . . . . (end of section **Experiments on the UML**) <<Contents | End>>

Review Questions

1. What does "UML" stand for and why do we learn it?
2. Describe the four(4) ways that people use the UML in software development.
3. Draw a simple class called "Circle" with private data member "radius" of type double and an operator called "area()" that returns a double.
4. Draw an object of class Circle with radius 4.
5. Draw an activity diagram of how you make a peanut butter and jelly sandwich.
6. Draw a diagram that shows that a Cat has four Legs and one Tail.
7. How many types of diagram are there in the UML?

. . . . . (end of section **Review Questions**) <<Contents | End>>

. . . . . (end of section **A Beginners Guide to The Unified Modeling Language (UML)**) <<Contents | End>>

Abbreviations

1. **Algorithm**::=*A precise description of a series of steps to attain a goal*, [**Algorithm**] (Wikipedia).
2. **Class**::=*A description of a type of object that includes the data it knows and the functions it can execute*.
3. **Function**::programming=*A selfcontained and named piece of program that knows how to do something*.
4. **Gnu**::="Gnu's Not Unix", a long running open source project that supplies a very popular and free C++ compiler.
5. **OOP**::="Object-Oriented Programming", Current paradigm for programming.
6. **Semantics**::=*Rules determining the meaning of correct statements in a language*.
7. **SP**::="Structured Programming", a previous paradigm for programming.
8. **Syntax**::=*The rules determining the correctness and structure of statements in a language*, grammar.
9. **Q**::software="A program I wrote to make software easier to develop",
10. **TBA**::="To Be Announced", something I should do.
11. **TBD**::="To Be Done", something you have to do.
12. **UML**::="Unified Modeling Language", industry standard design and documentation diagrams.
13. **void**::C++Keyword="Indicates a function that has no return".

End