BUCKNELL UNIVERSITY
CSCI 204 – INTRODUCTION TO UML DIAGRAMS IN DIA
BY KATIE HEISE

What is Dia?

Dia is a drawing program that can be used to create diagrams. Dia includes a variety of basic tools like lines and boxes and can load "sheets" – collections of tools that are used in a specific type of diagram. Most diagram objects have handles to which lines can be connected to form graph structures. When the objects are moved or resized, the connections will follow their respective objects. In this course, we will concentrate on working with UML class boxes.

What is UML?

Unified Modeling Language. UML diagrams are class diagrams that allow documentation of data members, member functions, interfaces, and relationships between classes. UML is one way to represent various views of a piece of software – the user's or client's view or the source code, for example. The diagrams provide a logical structure or outline for writing code.

Starting Dia For UML Diagrams on the Sun Workstations

1. Type 'dia &' at the command line prompt and press Enter. The main window should appear after an introductory title dialog.
2. Select **UML** from the menu in the middle of the Dia window. A collection of tools for working with UML will appear below the menu. (See Figure 1 at right.)
3. *To open an existing file*, choose **Open** from the **File** menu. Double click the desired file in the Open Diagram dialog or select the file and click **OK**. Skip steps 4 and 5.
4. *Create a new diagram* by clicking on **New** from the **File** menu. You will see an empty grid.
5. Select the **Create a class** icon. Then, left click on the grid. Your screen should resemble Figure 2.

Figure 1

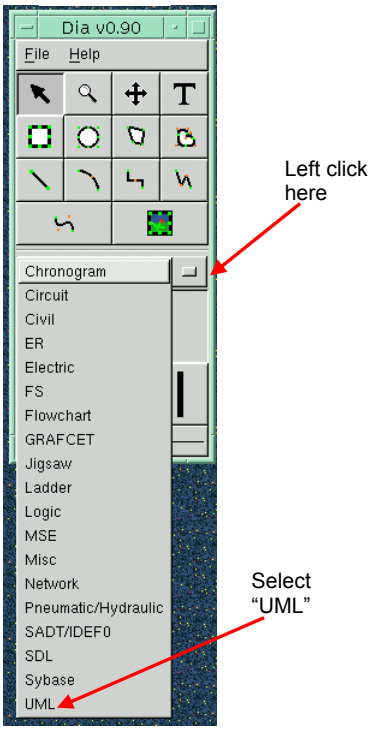
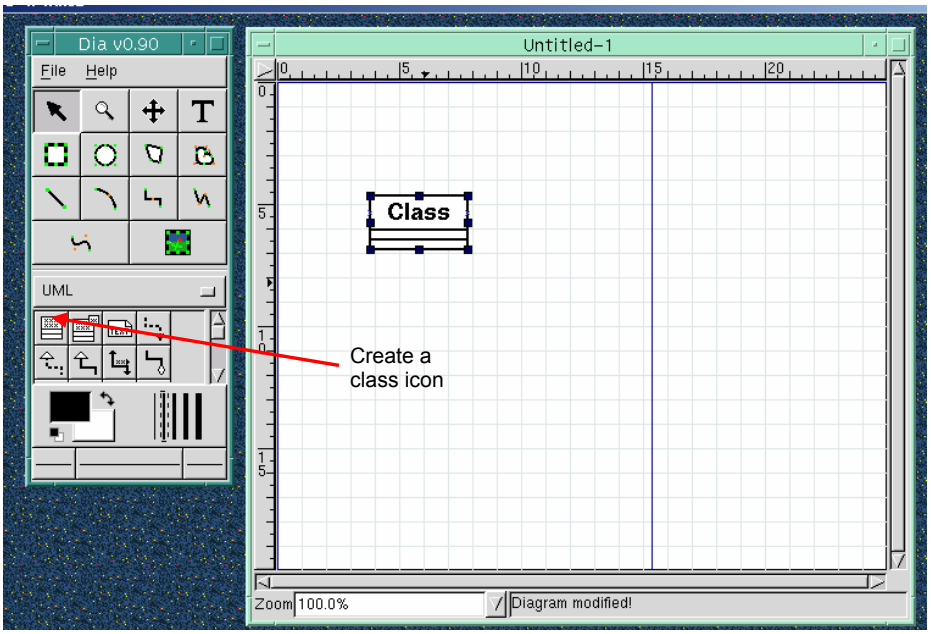


Figure 2



Working With UML Diagrams

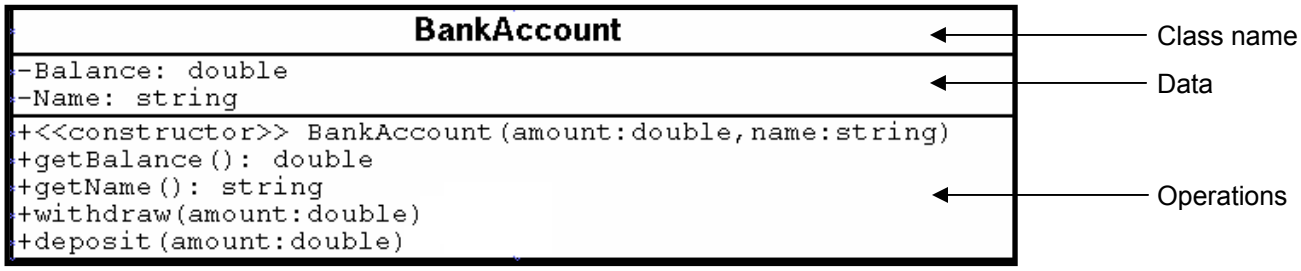
General notes: To access menus while working in the grid window, right click on the grid or on the object of interest. Don't forget to save using **File → Save** (CTRL-S) or **File → Save As…** (CTRL-W). Open an existing file from the Dia main window or from the grid by selecting **File → Open** (CTRL-O).

How do I edit my class?

To set the fields in your class box, select the box by left clicking. Double click within the class box to bring up the Properties window (or right click to bring up the menu and select **Dialogs → Properties**). See the BankAccount example on the following pages for specific information on creating and editing meaningful class diagrams using the Properties window.

How are classes organized?

Class diagrams are boxes divided into three sections. The class name appears at the top of the box. The data contained by each object of the class is found in the middle section, and the operations or functions compose the final section.



The class diagram above provides a great deal of information about the BankAccount class:

- A BankAccount object has two attributes, Balance and Name.
  - Balance is a double.
  - Name is a string.
- BankAccounts may be manipulated by several operations:
  - The constructor returns a new instance of BankAccount and takes two arguments – a double and a string.
  - getBalance returns a double and takes no arguments.
  - getName returns a string and takes no arguments.
  - withdraw takes a double argument.
  - deposit takes a double argument.

Note that instead of the C++ `void withdraw(double amount)`, for example, UML uses the syntax `withdraw(amount:double)`. void is not used in UML. '+' indicates that a data member or member function is public; '-' indicates that it is private.

Translating the class diagram into C++ yields the following code:

```
class BankAccount {
   private:
      double Balance;
      string Name;

   public:
      BankAccount(double amount, string name);  // constructor
      double getBalance();
      string getName();
      void withdraw(double amount);
      void deposit(double amount);
};
```

Example – Creating the BankAccount Class

Double click inside a new class box to open the Properties window.

Figure 3 – Step 1 of BankAccount Specification

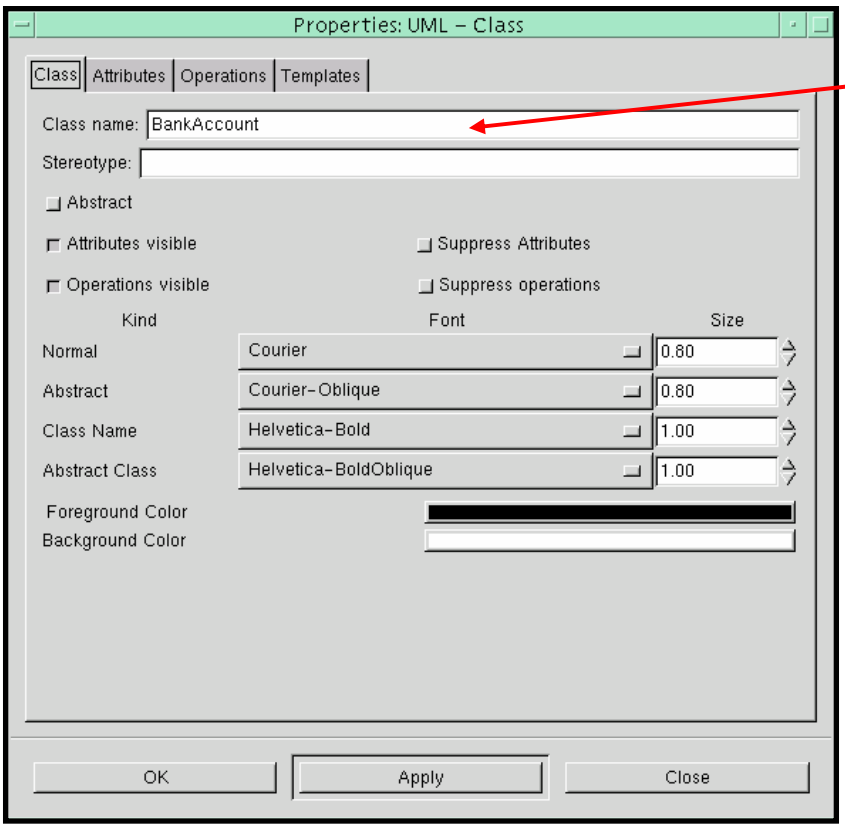


Figure 4 – Step 2 of BankAccount Specification – Adding an Attribute

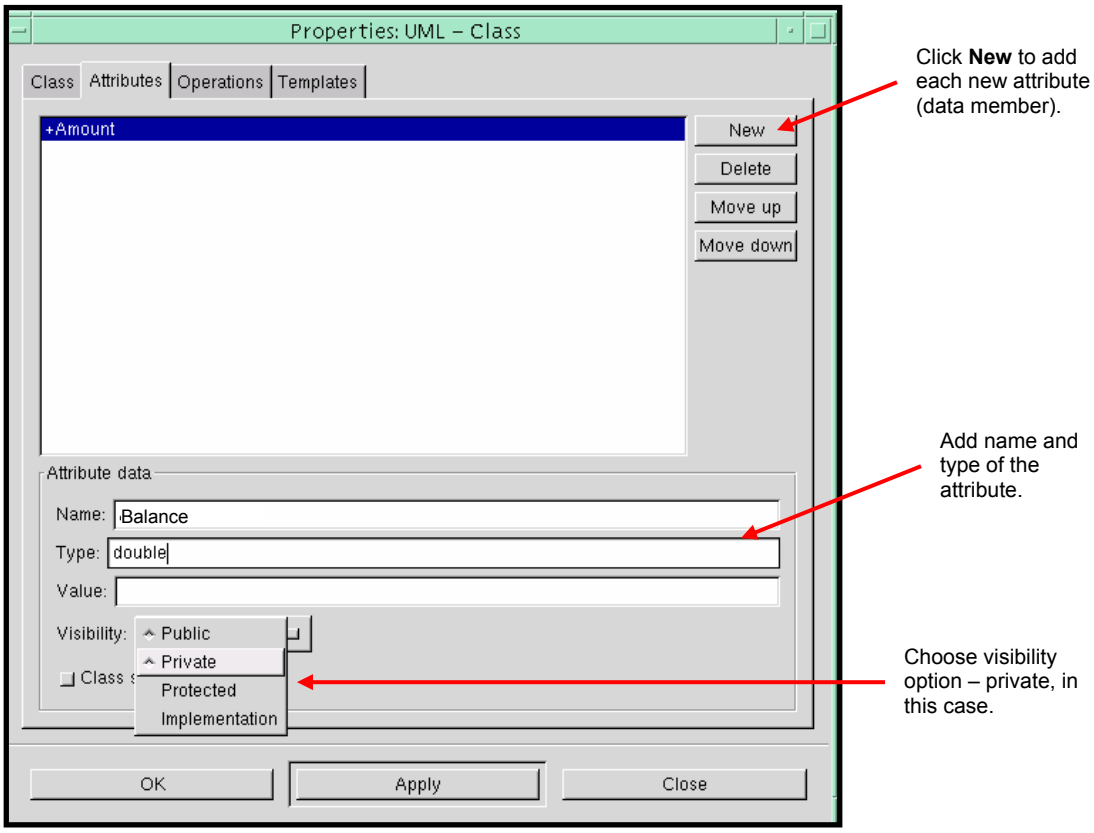
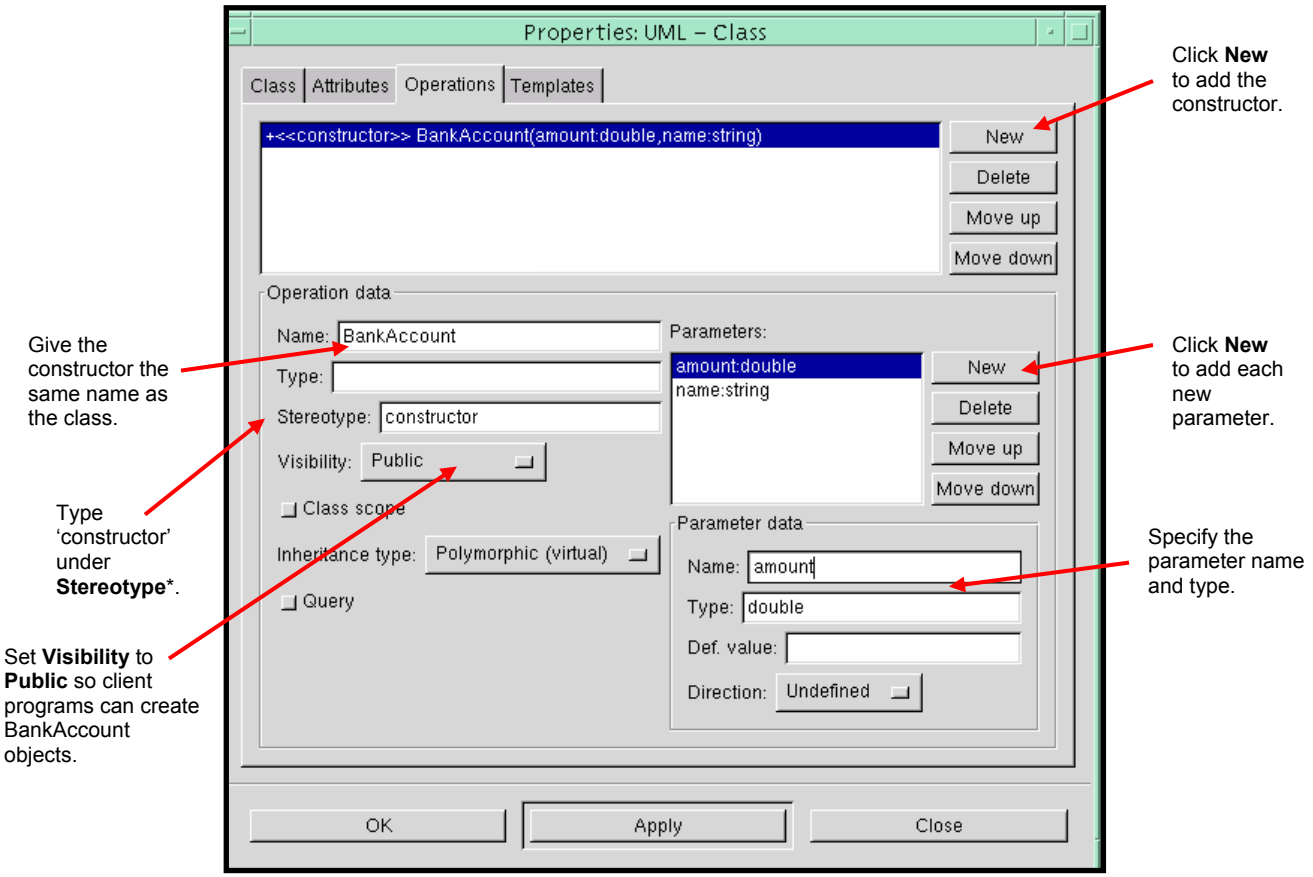
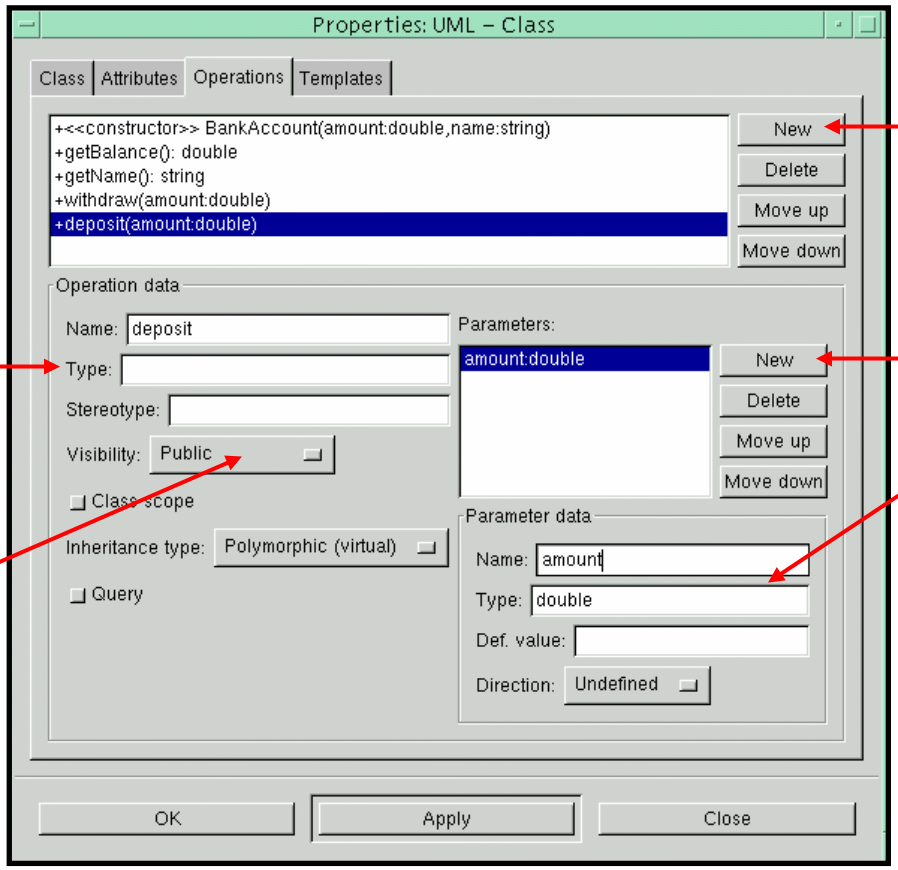


Figure 5 – Step 3 of BankAccount Specification – Adding the Constructor



* In UML, constructors do not have to share the name of the class. Constructors are identified with the stereotype <<constructor>>. When a stereotype – anything of the form <<…>> – is added to an aspect of a UML diagram, the meaning of that aspect becomes more specialized. In the case of <<constructor>>, the operation is forced to create and initialize a new object of the appropriate type. A constructor should not be documented as "returning" a type; it instead creates a new object.

Figure 6 – Step 4 of BankAccount Specification – Adding Other Operations



First, click **New** to add each new function.

Return type of the function (leave blank if `void`).

If the function has parameters, while the function name is highlighted in the list at the top of the window, click **New** to add each new parameter. Specify the name and type of the parameter.

Set visibility. All member functions are public in this example.

To *edit the class's attributes or operations*, click on the appropriate tab of the Properties window. Left click on an item in the list at the top of the window to select the attribute or operation to be modified. To modify a parameter, select the function and then click on the parameter. Blue highlighting indicates the selected component(s).

* For **const** functions or const or const & (reference) parameters, enter '`const`' before the type name in the operation's or parameter's Type text box (on the Operations tab of the Properties window) and, if appropriate, '`&`' after the type name. Add const to data members in the same way under the Attributes tab of the Properties window.

Printing a Diagram

Unless instructed otherwise, Dia tends to print large diagrams that extend over several pages. The dark blue lines on the grid mark the boundaries of each printed page. To solve this problem, try one of the following:

- Select **File → Page Setup…** by right clicking on the grid and select the **Fit to:** option under Scaling. The first value is the number of pages the diagram(s) on the grid will occupy horizontally, and the second value is the number of pages the diagram(s) will occupy vertically. This option will usually be set to **1 by 1**.
- Select **File → Page Setup…** as described above and set **Scale** under Scaling to the desired value. 30% often works well.

Print the diagram with the **File → Print Diagram…** (CTRL-P) command. Type the appropriate printer command in the text box ('`a2ps -Plwc`' for Dana 231 and '`a2ps -Plw3`' for Dana 320). Note that you may also print to a file.

Modeling Relationships

Associations

Before writing code, it helps to produce a model of the concepts involved, which can lead to easy identification of the necessary classes, functions, and variables. The simplest way to do this is to create a class box for each of the obvious classes in the problem and add associations between them. Click on the Association icon (shown at right) and draw a link between two classes on the grid by left clicking and dragging. Figure 8 shows the classes expected in the initial design of software that models a university. Since so many classes are involved, the details of operations and attributes are omitted, leaving only the class name in each class box. To achieve this effect, double click on a class box to open the Properties window, go to the Class tab, and clear the check boxes for **Attributes visible** and **Operations visible**.

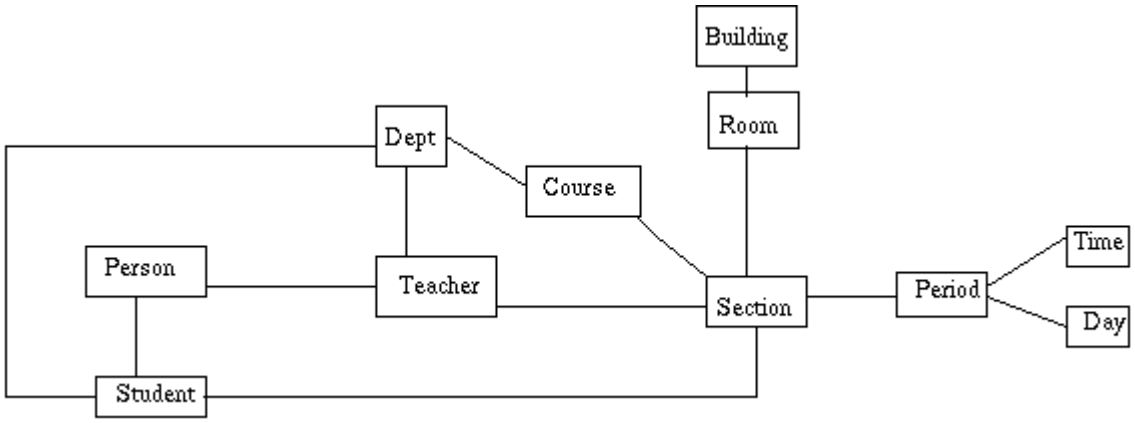


Figure 8 – Associations
Modeling a small campus. Taken from http://www.csci.csusb.edu/dick/samples/uml1.html.

Associations can be modified to provide more specific information. Double click on the association (the dark line connecting two classes) to bring up the Association window. All fields are optional. Only one of the four Aggregate and Composition boxes may be checked at any given time. Dependencies and composition will be covered in more detail in the next few sections of this tutorial.

See Figure 10 on the next page for a revised version of Figure 8. Association and role names and multiplicities have been added.

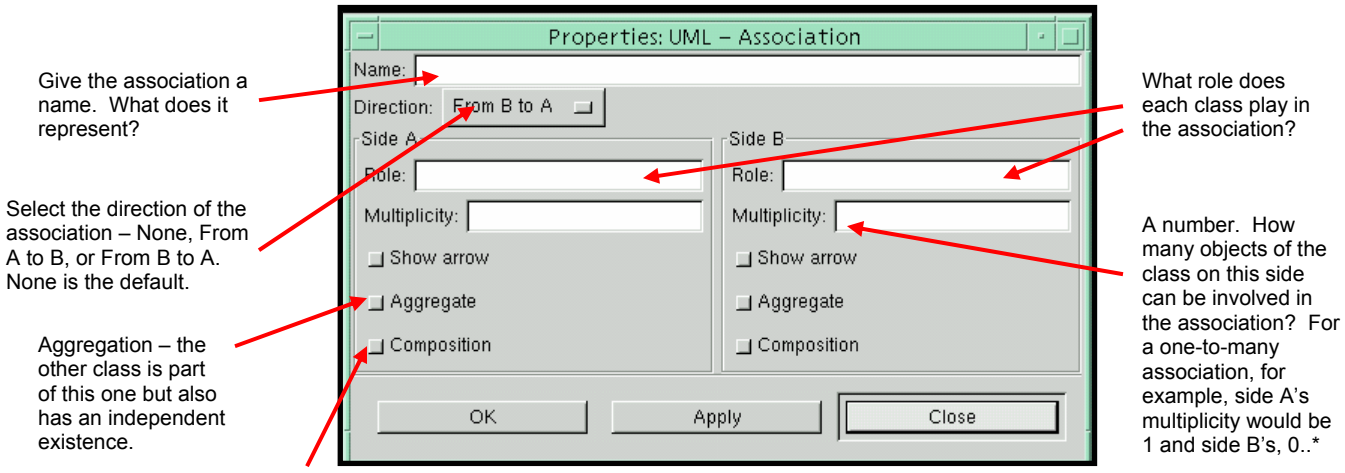


Figure 9 – The Association Window

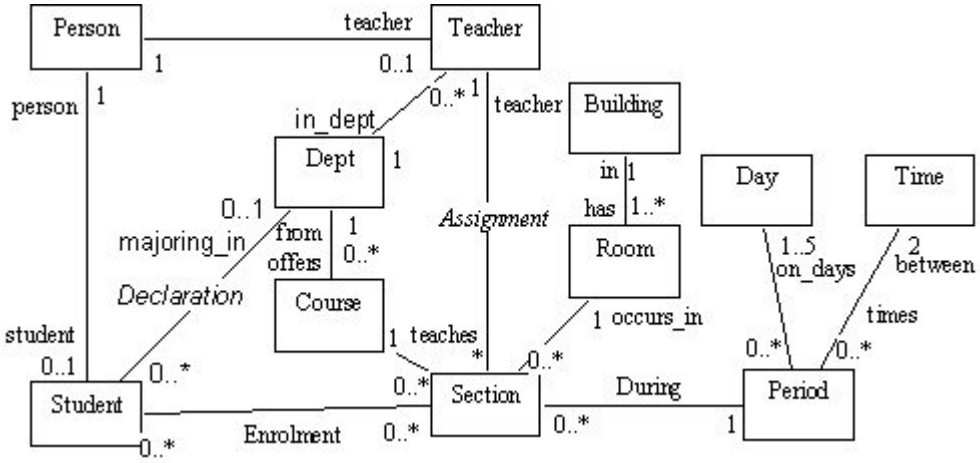


Figure 10 – Modeling a Small College
Association names and multiplicities have been added. Taken from
http://www.csci.csusb.edu/dick/samples/uml1.html.

Dependencies

When one class makes use of another in some way, the relationship is represented by drawing a dashed arrow from the client class to the class providing the service. Identifying dependencies is simple. If the code of class A contains the name of another class in the diagram, class B, there should be a dependency drawn from A to B if no other connection exists between them. Regardless of the number of uses of class B within A, only one dependency arrow is necessary. The Create a dependency icon is shown at right.

Figure 11
Each function in class A individually gives rise to a dependency on B. This figure demonstrates different ways to incur a dependency.

```
class A {
    public:
        B getB();
        void processB(B b);
        void bInImplementation();
};

void A::bInImplementation() {
    B b;
    …
}
```

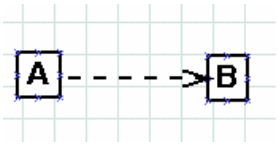


Figure 12
Class A uses class B.
Class A depends on class B.

In the following cases, do *not* use a dashed arrow dependency. Dia and UML provide more specific tools to represent such relationships:

- One class has a data member of the other's type – *Composition* (See the following section.)
- One class has a pointer or reference to the other – *Aggregation* (See http://www.csci.csusb.edu/cs202/uml1b.html#Aggregation.)

- One class is derived from the other – *Generalization* ("Every A is also a B") (See http://www.csci.csusb.edu/cs202/uml1b.html#Generalization.)

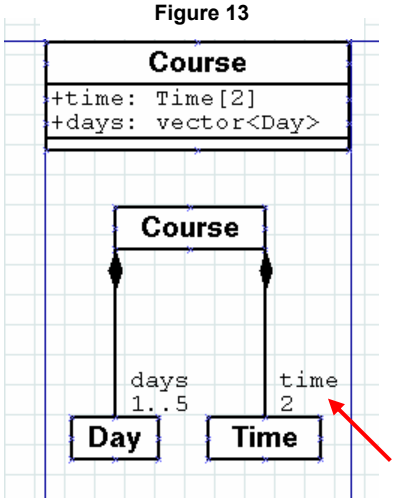
- One class implements the other – *Abstraction* (See http://www.csci.csusb.edu/cs202/uml1b.html - Abstraction and Implementation.)

Composition

In general terms, composition is a “has a” relationship. For example, a person object would have arms, legs, and eyes as parts. A car has an engine, a chassis, and a set of tires. C++ structs are prime examples of something that is, at heart, a collection of parts. And data members are parts of the state of each object of a class.

UML has special notation to show a composition relationship. Add a general association connecting the two classes (discussed above) and select the **Composition** check box for the side with the class that has a component of the other class or type.

Figure 13



UML uses a black diamond to indicate that an item is stored within another piece of data *and* to indicate that a class of objects has responsibility (temporary or permanent) for the existence of one or more objects of the class at the other end of the association. When an item is stored inside an object, the creation or deletion of the object will automatically cause the creation or deletion of its parts.

In Figure 13, for example, the creation of Course object causes a vector of Day objects and an array of Time objects to be created. These data structures are part of the Course object’s state and will be eliminated when the Course is deleted or falls out of scope.

Dia / UML Resources

Feel free to experiment with Dia and with the more advanced options for UML diagrams. The following websites may help:

- http://www.lysator.liu.se/~alla/dia/diatut/all/all.html
  Dia tutorial – installation, the basics, and advanced formatting
- http://www.rational.com/uml/
  UML resource center and documentation; from the developers
- http://www.csci.csusb.edu/dick/cs202/lab02.html
  Very good resource; follow links to find information on basic and more advanced topics
- http://www.csci.csusb.edu/dick/samples/uml1.html
  Simple modeling techniques; relationships between C++ and UML