StarUML Tutorial*

Stephen Wong

Abstract

StarUML is a tool to create class diagrams and other types of diagrams in the Unified Modeling Language (UML). This module is a short tutorial on using StarUML to create class diagrams in Java.

StarUML

StarUML (SU) is a tool to create UML class diagrams and automatically generate Java "stub code". SU can also reverse engineer Java source/byte code to produce the corresponding UML diagrams.

In this tutorial, we will use SU to design a Pizza program. Perform the following steps to create the UML diagrams shown below. SU will generate code that reflects the class structure, but not the specific actions on any objects. For that, after creating the diagram using SU, you'll edit the resulting stub code to add the rest of the functionality to the code, filling in what each method should do.

*Version 1.1: Sep 10, 2007 12:37 pm -0500
†http://creativecommons.org/licenses/by/2.0/

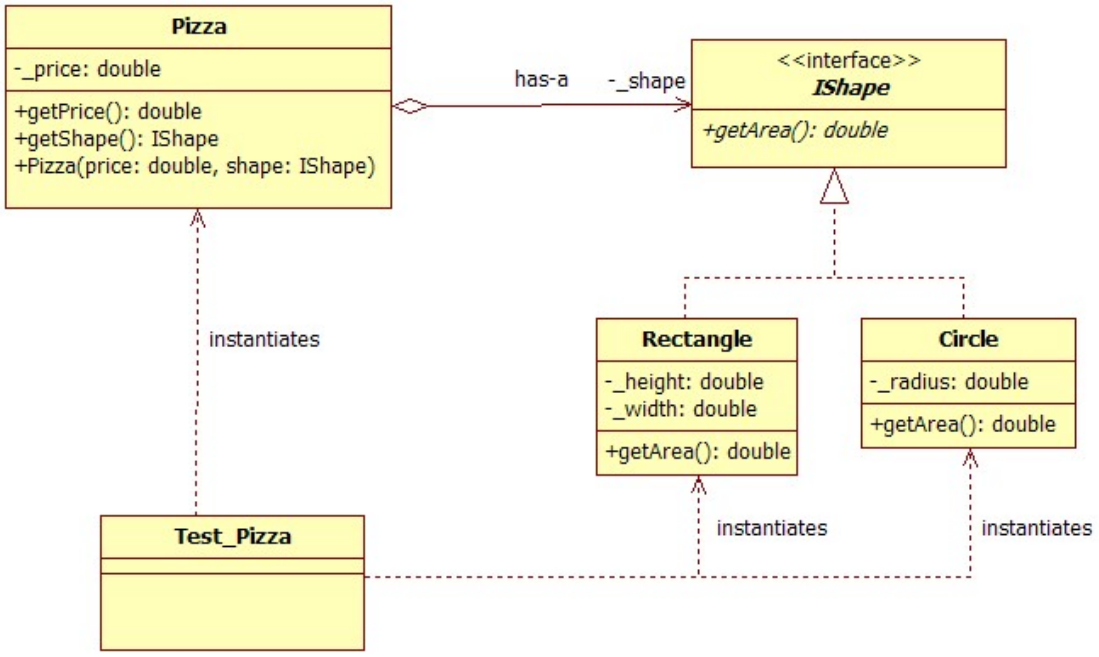
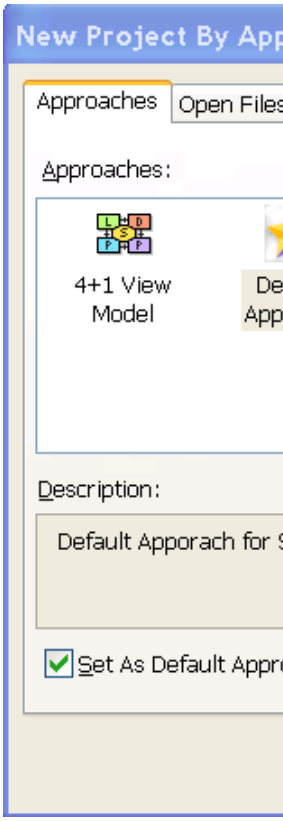


Figure 1

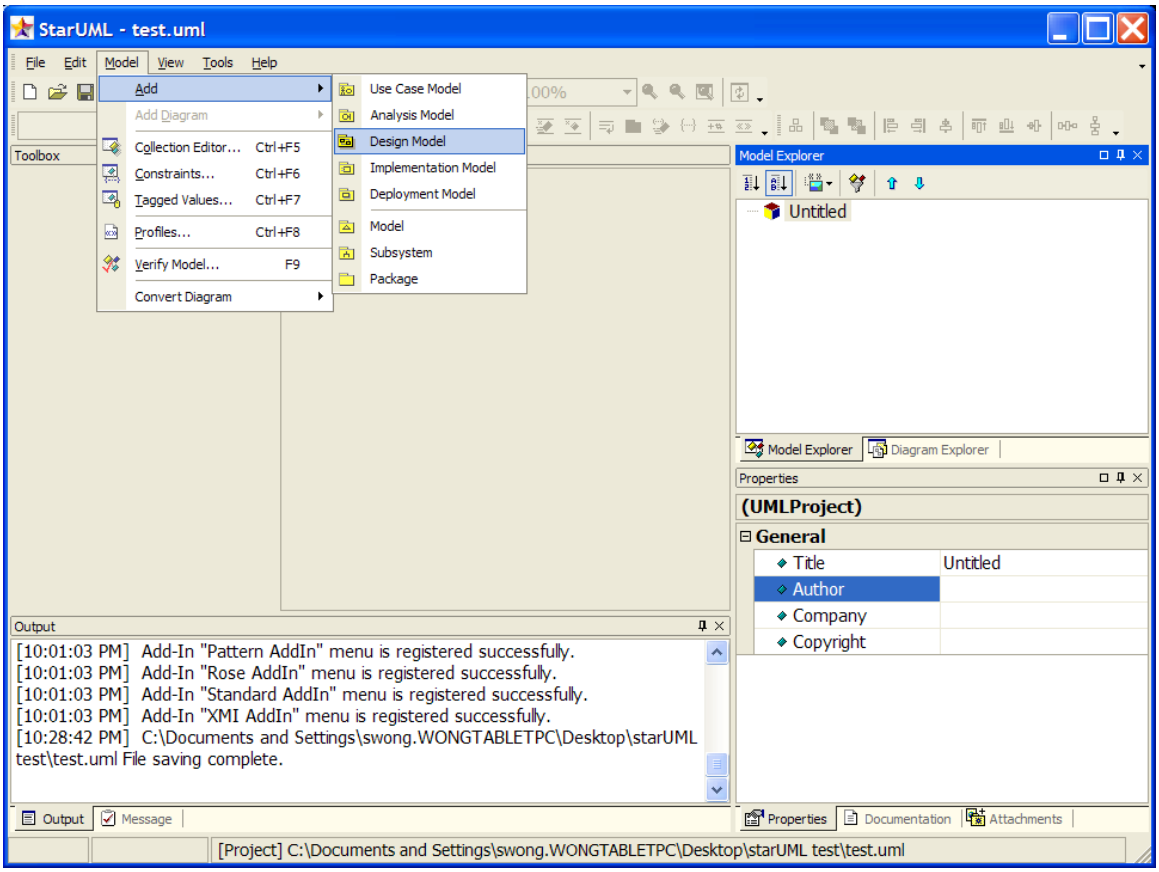
1. Installation: To begin, we must first install the software we will be using. if it is not already installed. The package, StarUML, is open source software, licensed under the GPL (GNU Public License)[1] , and is freely available for download from its homepage[2] . And here is a direct link to the package itself[3] .
2. Once StarUML ("SU") is installed, start the program.
3. After starting SU, a template box titled "New Project by Approach" may be present: if it is, select

[1] http://www.gnu.org/licenses/licenses.html#GPL
[2] http://staruml.sourceforge.net/
[3] http://downloads.sourceforge.net/staruml/staruml-5.0-with-cm.exe



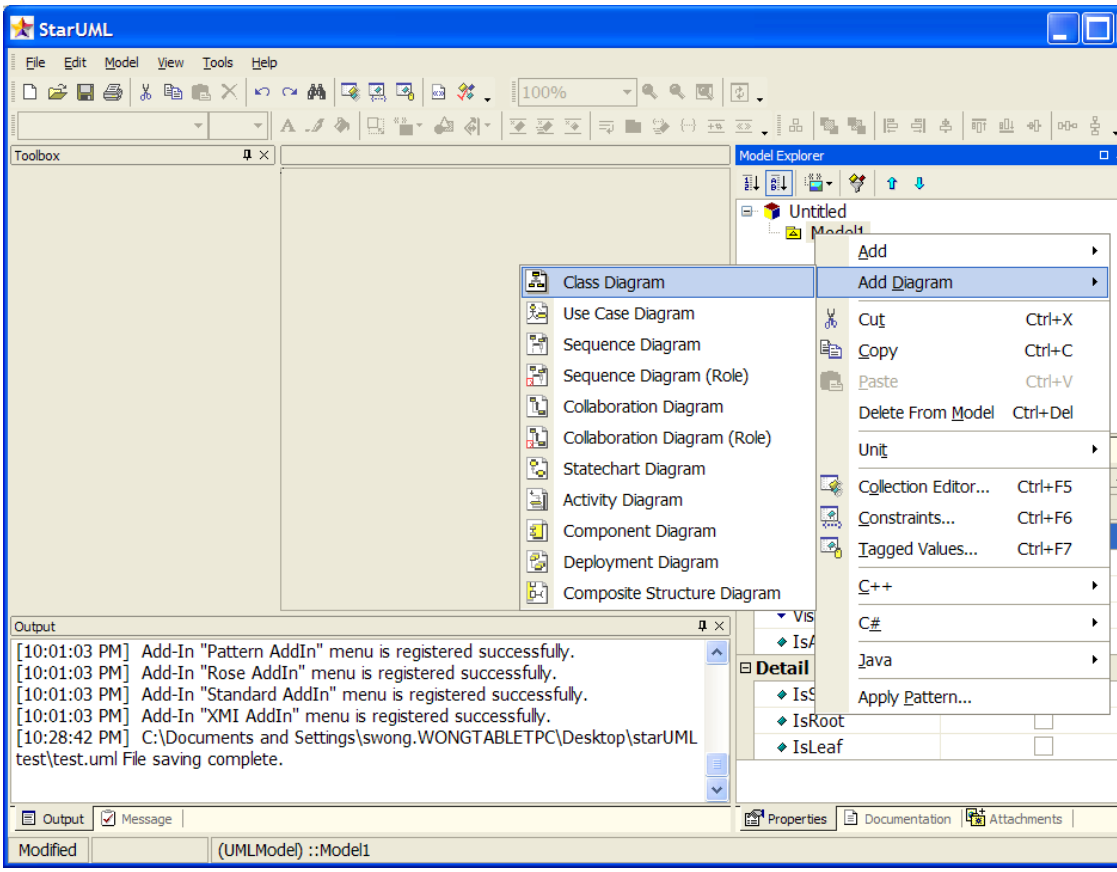
"Empty Project" and press "Ok". It is suggested that you uncheck "Set As Default Approach".

4. On the "Model Explorer" pane on the upper right side, select the (as-yet) "Untitled" model.
5. Either on the main menu under "Model", or by right-clicking the selected model, got to "Add/Design



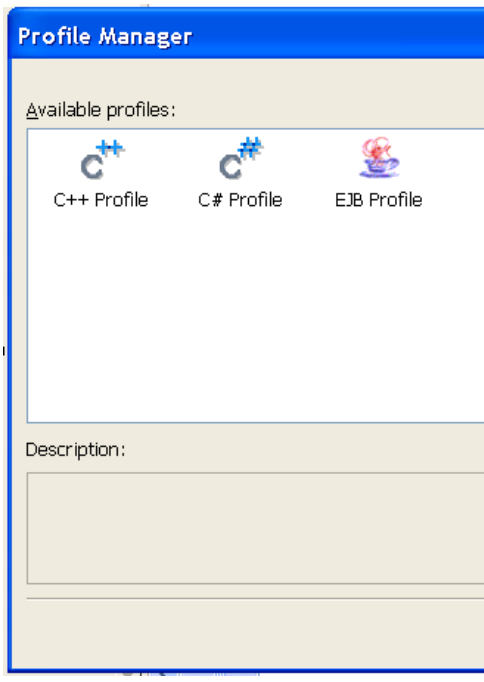
Model"

6. Either on the main menu under "Model", or by right-clicking the selected model, got to "Add Dia-

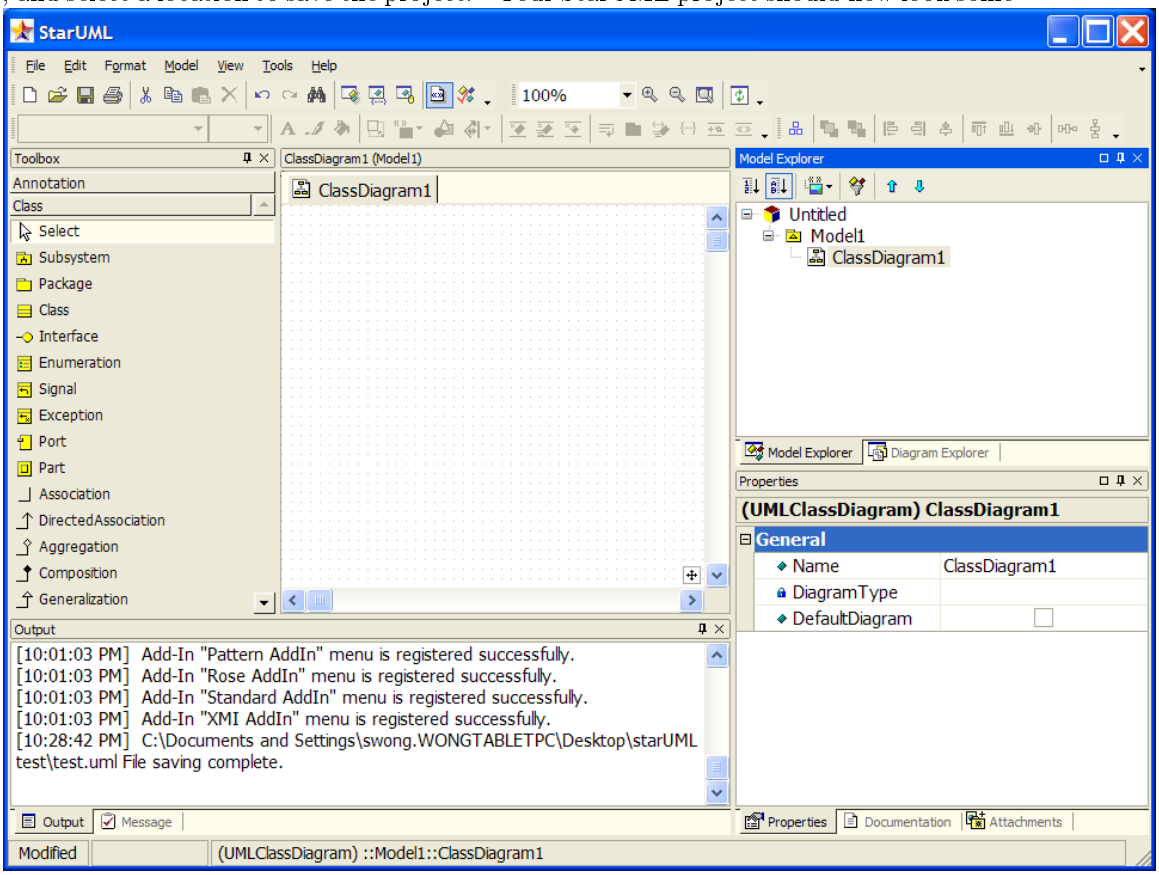


gram/Class Diagram":

7. Go to "Model/Profile..." to set the "profile" used by the project, which determines which symbols and conventions will be in use. Be sure to include the "Java Profile" in the project.



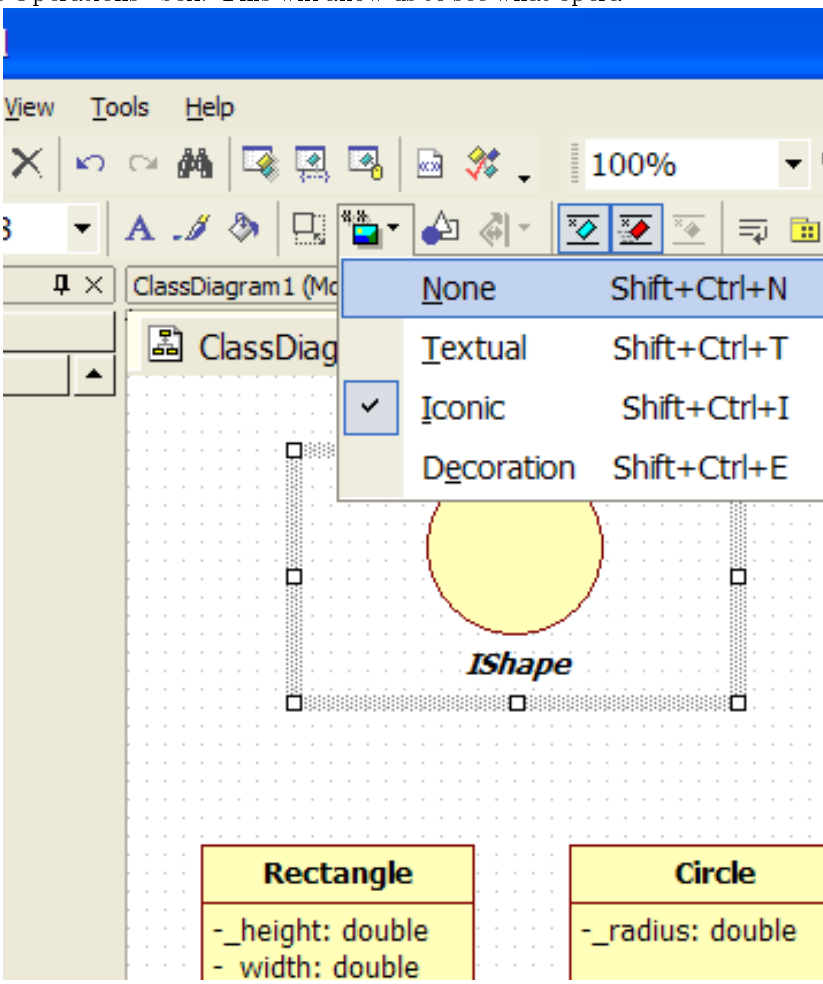
8. Save the project now so you do not lose any progress if some problem arises. From the "File" menu, choose "Save", and select a location to save the project. Your StarUML project should now look something like this:



9. Now to begin actually creating the diagram, from the "Toolbox" which starts by default on the left side of the screen, select the "Class" icon, and left-click somewhere on the diagram window. This should create a new class with a generic name. Rename the class to Circle by double clicking on the name.
10. Add an "Attribute" (or field) to Circle by right-clicking the object on the diagram, expanding the "Add" menu, and pressing the green "Attribute" button. Type in the desire name of the field, "_radius".
    - Specify the data type in the Properties panel (lower right side of window) by typing double in the "Type" slot.
    - Internal data of a class (field/attributes) are always private because they are strictly for personal use by the class to help it determine its behavior. So, in the Properties panel for the _radius field, select PRIVATE for its Visibility.
11. Repeat the same process to create a class called Rectangle with private _width and _height fields of type double. You may notice using the "Model Explorer" on the right is faster to add these, but do however note that adding the classes and interfaces themselves in this toolbox (instead of using the toolbox on the left and clicking on the palette to create the object) will not create the objects in the diagram. If you choose to use the "Model Explorer", the area we will be interested in is visible after expanding the "Design Model" group.
12. Create an interface called IShape
    - From the toolbox, choose "Interface" and click somewhere on the palette. Rename the generic

name to IShape. Select the interface by left-clicking the item after it is created.

- On the top toolbar, select the dropdown "Stereotype Display" and change the value to "None". This will change the previous circular shape into a rectangular shape.
- Also on the toolbar, de-select the "Suppress Operations" box. This will allow us to see what operations the interface has in the diagram view.



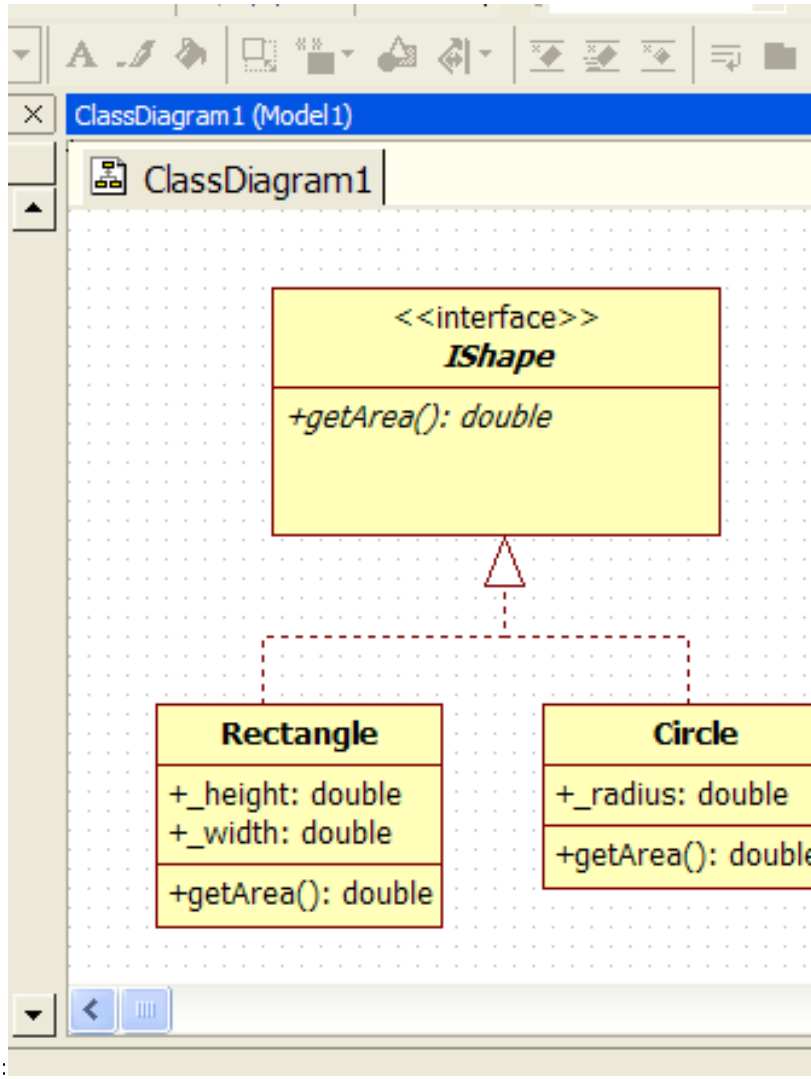
- Add a getArea method of type double to the IShape interface.
  - This can be accomplished by right clicking the interface, expanding the add menu, and pressing the red "Operation" button. Enter the name as: getArea.
  - To set the return type, expand IShape in the "Model Explorer", right click the getArea method you just created, and select "Add Parameter". In the "Properties" box, change the parameter's name to nothing, "", change the "DirectionKind" to "RETURN", and change the "Type" to double.
- On both the IShape interface itself as well as its getArea method, check the IsAbstract box in the Property pane. This will make their titles appear as italics, as per the UML standard for interfaces and other purely abstract entities.

13. Make Circle and Rectangle implement IShape by selecting the "Realization" arrow from the toolbox, clicking on Circle and dragging the line to IShape. Repeat the same process for Rectangle. This is adding the relationship that Circle and Rectangle will implement the IShape interface.
    - To make the connector line make nice right-angle bends, right-click the line and select "For-

mat/Line Style/Rectilinear". You can make your diagram look cleaner by simply laying arrowheads that point to the same place right on top of each other, making it it look as if there is only one arrowhead.

14. Since the Circle and Rectangle class both implement the IShape interface, they must have the same behaviors (methods) as IShape.
    - In the Model Explorer pane, copy the getArea method (Ctrl-C or right-click and select Copy) from IShape to both Circle and Rectangle.
    - The implemented methods in the Circle and Rectangle classes are not abstract, but concrete because they actually perform some particular action (i.e. calculate the area for a circle and rectangle respectively). So, uncheck the IsAbstract box for those methods.

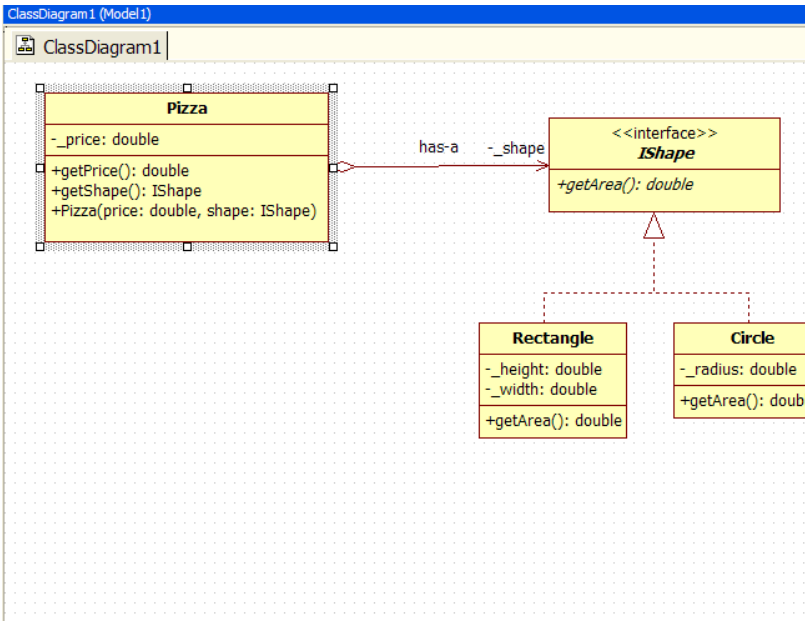
15. Your diagram should now look something like this:



16. Add a class called Pizza.
    - Add a private _price field of type double.
    - Add a public getPrice operation that returns type double.
17. To make Pizza reference an IShape, select class Pizza.

- Select the "DirectedAssociation" arrow in the toolbox, click on Pizza, and drag to IShape.
  - Now select the arrow, and in the "Properties" box on the right, change the name to "has-a", change "End1.Aggregation" to "AGGREGATE" (this is a formal diagrammatic statement that a pizza is made up, i.e. "aggregated", with another object, a shape object).
  - Change the "End2.Name" to _shape. This will automatically add a private field called _shape of type IShape to Pizza.
  - change the End2.Visibility to PRIVATE.
- Create a "gettor" method (Routine) for _shape called getShape that returns IShape. That is, create an operation called getShape that returns IShape.

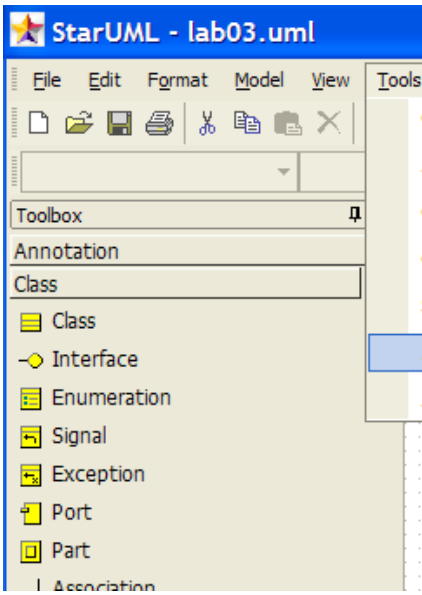
18. Constructors are special pieces of code used to initialize an instance of a class when it comes into existence.
    - To add a constructor for Pizza, right click on Pizza, expand the "Add" menu, and select "Operation". From here, add a normal operation as usual, with input parameters double price and IShape shape.
      - Adding an input parameter is just like adding an output parameter for the return type earlier, except you specify the desired parameter name, such as price and shape, and the appropriate data type.
    - Add a Circle constructor with parameter double radius.
    - Add a Rectangle constructor with parameters double width and double height.
19. Your diagram should now look something like this:



20. To illustrate one more type of UML class diagram feature, add another class to your diagram called "Test_Pizza". This would be a class that uses the Pizza and IShape-derived classes, say, for testing purposes.
    - Dependency lines help show relations between classes that occur more dynamically. For instance, one class may instantiate another class but not hold permanent a reference to it by using a field. Or a class's method may take another class as an input parameter, retaining a reference to it only for the duration of the execution of that method.
    - Add dependencies between different classes by selecting the "Dependency" arrow from the toolbox, selecting a dependent class, and dragging the arrow to the class it is dependent upon. In this example, Test_Pizza "depends" on Pizza, Circle, and Rectangle because it instantiates them.

- Enter a label for a dependency by changing the "Name" property in the Properties box or by double-clicking the dependency line. Typically when one class instantiates another class, we label the dependency line "instantiates" (surprise, surprise!).
- You can move the label of the dependency line around to a more aesthetic location by selecting the label on the diagram and dragging it.
- Dependencies have no effect on code generation.

21. Your diagram should now look like the diagram at the top of this web page.
22. Feel free to make other modifications to your diagram. You can drag your class diagrams around and bend the arrows in many different ways (to make the arrows rectilinear, select an arrow, right click it, expand format, expand Line Style, and select Rectilinear). You just have to experiment with the tool to get to know it.
23. In the File menu, select Save. SU uses a single project file for all the information, so you should have only 1 file generated currently.
24. It will be useful to export diagrams to other formats, such as images. You can do this by selecting "Export Diagram" on the File menu and choosing an appropriate file type.
25. To generate the Java stub cod:



- Go to "Tools" on the main menu, expand "Java", and select "Generate Code".
- From this dialog box, select your model, probably named "Model1"and press "Next"
- Choose "Select All" to generate stub code for all the classes in your model/diagram and then press "Next".
- Select a valid output directory and select "Next"
- In the "Options Setup", be sure to check both "Generate the Documentation by JavaDoc" and "Generate empty JavaDoc". All other checkboxes should be unchecked. Then press "Next".
- StarUML will now generate the code stubs from your diagram. Click "Finish" to exit the dialog.
- You can now edit the generated stub code to add functionality to the application.

26. Now define what this program actually does, i.e., write the code to implement the methods described by your diagram.
    - Use DrJava to add code to the corresponding class .java file. The code would be the same as you wrote for HW02[4] . (Note: the code for Test_Pizza is best autogenerated by DrJava rather than created by hand in StarUML. We just showed it here for illustrative reasons.)

[4] http://www.owlnet.rice.edu/~comp201/07-spring/assignments/hw02

- Remember that the IShape getArea() method is abstract and so has no code body.
- Make sure you add comments to the code as shown in the sample code. The comments are written in "JavaDoc" style. You will learn more about JavaDoc in subsequent labs.

27. StarUML is also capable of creating a class diagram from existing Java code, what is referred to as "reverse engineering" the code. This is very useful when you have existing code you want to diagram or if you have modified code that was generated by StarUML by adding fields and methods and you thus want to update your diagrams to reflect those changes. The process of going back and forth between working on your code through a diagram and through a text editor such as DrJava, is called "round-trip engineering" and is the basic design process used in object-oriented programming.
    - To reverse engineer some existing code, go to the main menu bar and select "Tools/Java/Reverse Engineer...".
    - Select the directory that holds the Java (.java) files that you wish to use and click the "Add" or "Add All" button to include them in the reverse engineering process. Then click "Next(N)".
    - Select the model you wish to add the classes to, probably "Model1" then click "Next(N)".
    - In the Option Setup:
        - Be sure that the options for generating "public", "package", "protected" and "private" visibility are all checked (this is the default).
        - Also, by default, the radio button for "Create the field to the Attribute" should be selected.
        - Unless you want SU to create another, badly laid-out, class diagram for you showing all the classes in your model, uncheck the "Create Overview Diagram" box.
        - When you are done checking your options, click "Run(R)".
        - SU will now import the classes in the selected files into your model. Click "Finish" to exit the dialog when it is complete.
    - SU will add any imported classes into your model, but not your diagram. To add them to your diagram, simply drag them from the Model Explorer to your diagram.

Bug warning: As of version 5.0.2.1570 of StarUML, when a generalization line has been deleted from the diagram, it may not actually be deleted from the underlying model. This may cause erroneous code generation. To detect if there are excess relationships still lingering for a class, do the following:

1. Right-click the class and select "Collection Editor".
2. In the Collection Editor, select the "Relations" tab.
3. The Relations tab will show all the relations associated with the class, both those line pointing away from the class as well as those pointing towards it.
4. If there are more relations shown in the Relations tab than showing on the diagram, check to find out which one is in error.
5. To delete a relationship that is not visible on the class diagram, right click the relationship in the Relations tab and select "Delete".

More information on running StarUML can be found here[5] .

[5] http://staruml.sourceforge.net/docs/user-guide(en)/toc.html