HW-CD StarUML

This is an individual homework assignment. In this HW, you will learn about StarUML, a UML tool. Along the way, you will answer some questions.

Contents

[1 Introduction 1](#_Toc104280031)

[2 Configure for First Use 2](#_Toc104280032)

[3 Build Class Diagram 4](#_Toc104280033)

[4 Forward Engineer: Generate Code 8](#_Toc104280034)

[5 Reverse Engineer Code: Generate Diagram 10](#_Toc104280035)

[Appendix 1 Configuration 17](#_Toc104280036)

[Appendix 2 Resources & Sources 17](#_Toc104280037)

# Introduction

So far, I have found *StarUML* to be a very good tool for UML. The visual editor, for drawing a class diagram is fairly intuitive, which I can’t say for most of the other tools I have tried (Yatta-UML Lab, Visual Paradigm, UMP, Bouml, Modelio). It is a bit labor intensive, but to a lesser degree than the others. StarUML supports many UML diagrams: Class, Use Case, Component, Deployment, Sequence, Activity, State, *etc.*). However, here, we only consider the class diagram.

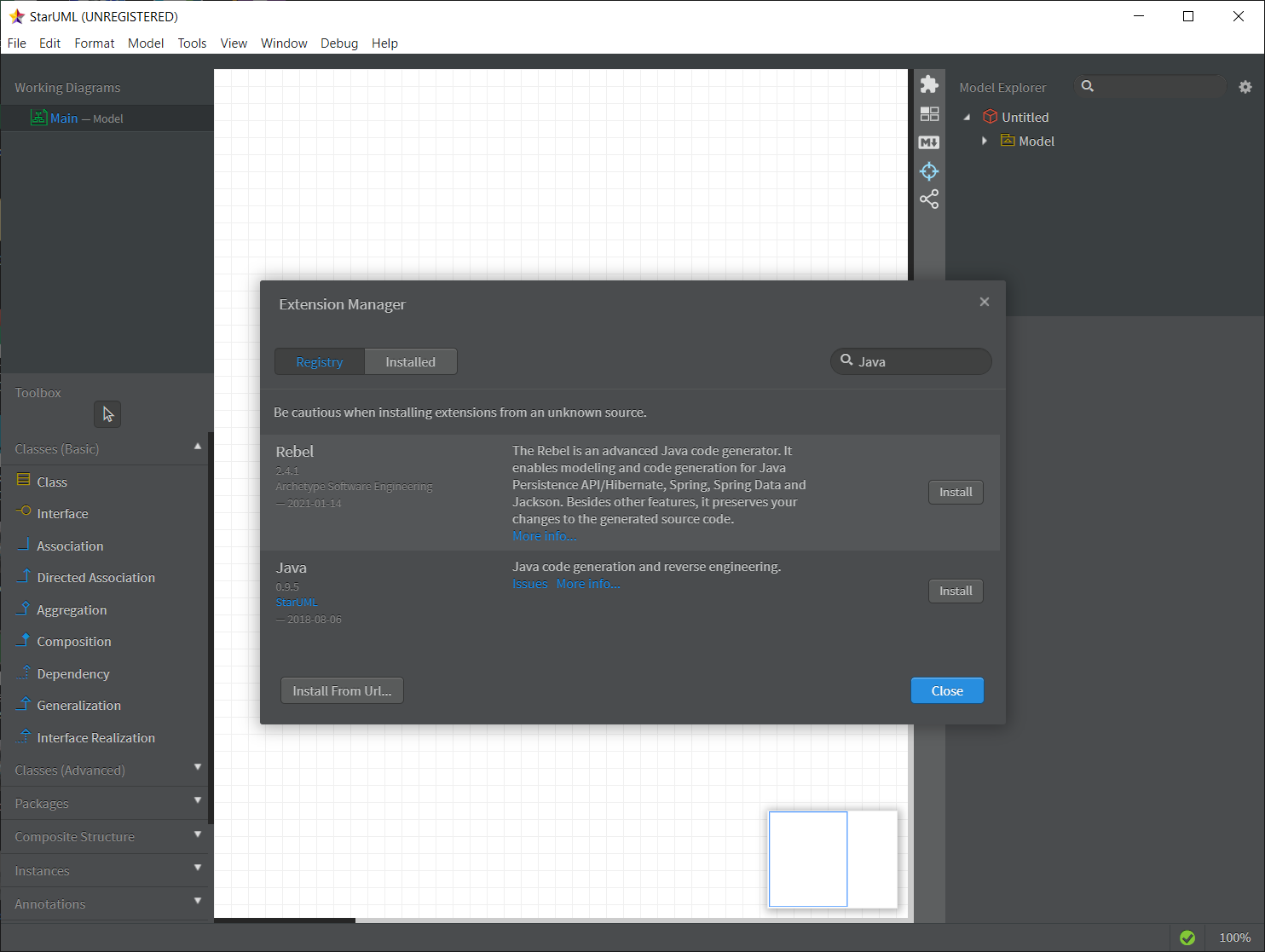
StarUML provides an open API so there are a number of plugins available: <https://staruml.io/extensions>. We will use three: (a) Java – which allows code generation and reverse engineering; (b) Generate Getters & Setters – when you add an instance variable, this will provide a way to automatically generate a getter and setter; (c) Merge Generalizations – when you have two or more subclasses, this allows all the separate generalization arrows to be merged into one.

The official documentation is pretty good: <https://docs.staruml.io/>. Although not necessary, if you want to get better insight on how StarUML could be used in practice, read the brief pages: Introduction, Basic Concepts, Managing Projects, Managing Diagrams, User Interface.

*Forward Engineering* refers to the ability to automatically convert a UML class diagram into code. *Reverse Engineering* refers to the ability to generate a UML class diagram from code. StarUML does both, and this homework/lab will show you how to do these things. In particular, the reverse engineering will be important for your project where you will need to generate various class diagrams of your code base.

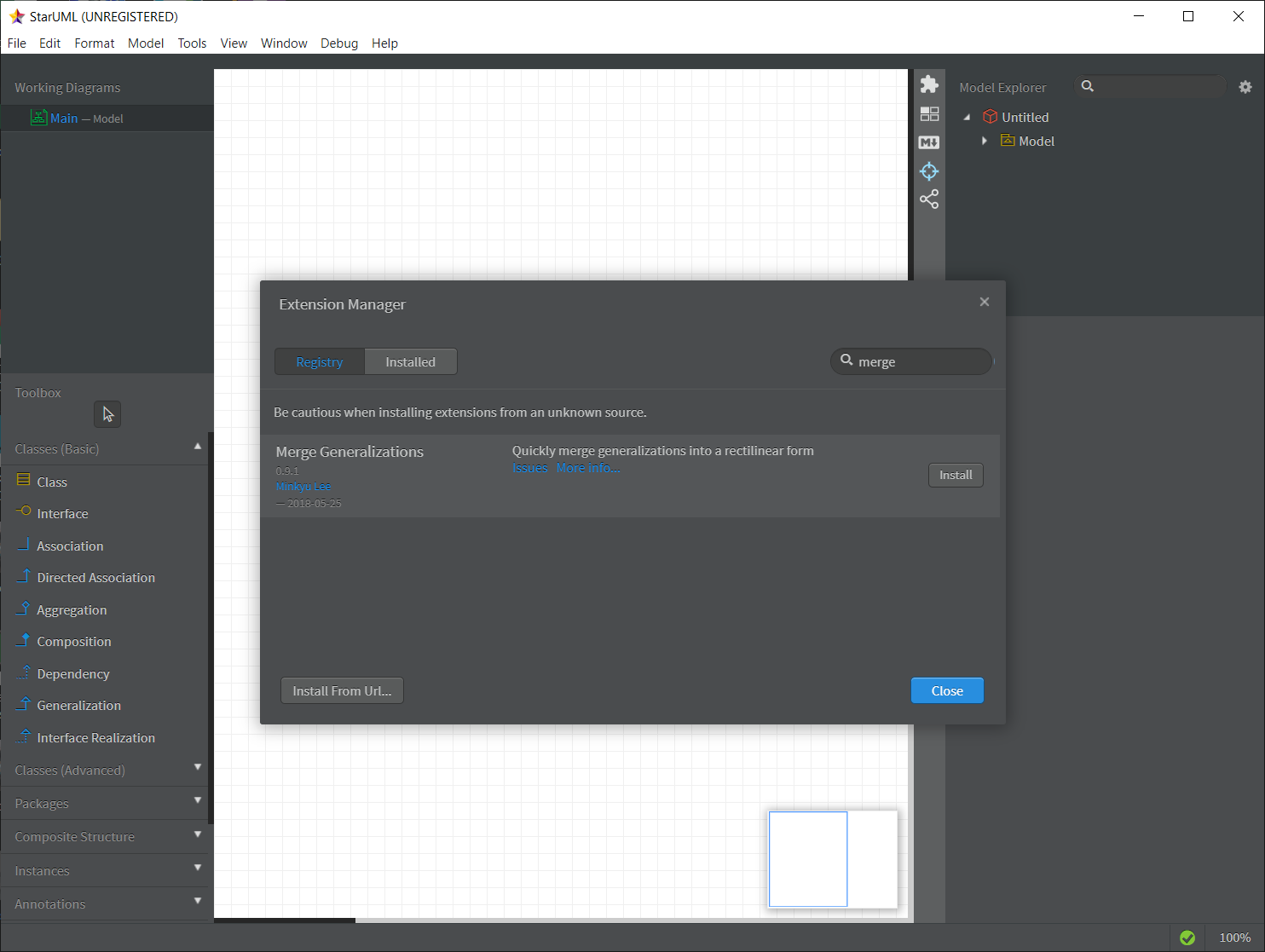
# Configure for First Use

1. Download and install StarUML: <https://staruml.io/>
2. Run StarUML, choose: Evaluate. A nag screen appears, but informs you that there is no time limit for the evaluation of the software.
3. Next, we install the three extensions mentioned earlier. Choose: Tools, Extension Manager and search for “Java”. Install this (the second one listed in the figure below). It will say it needs to Reload. Choose: Close instead, as we will reload when we have installed all three.



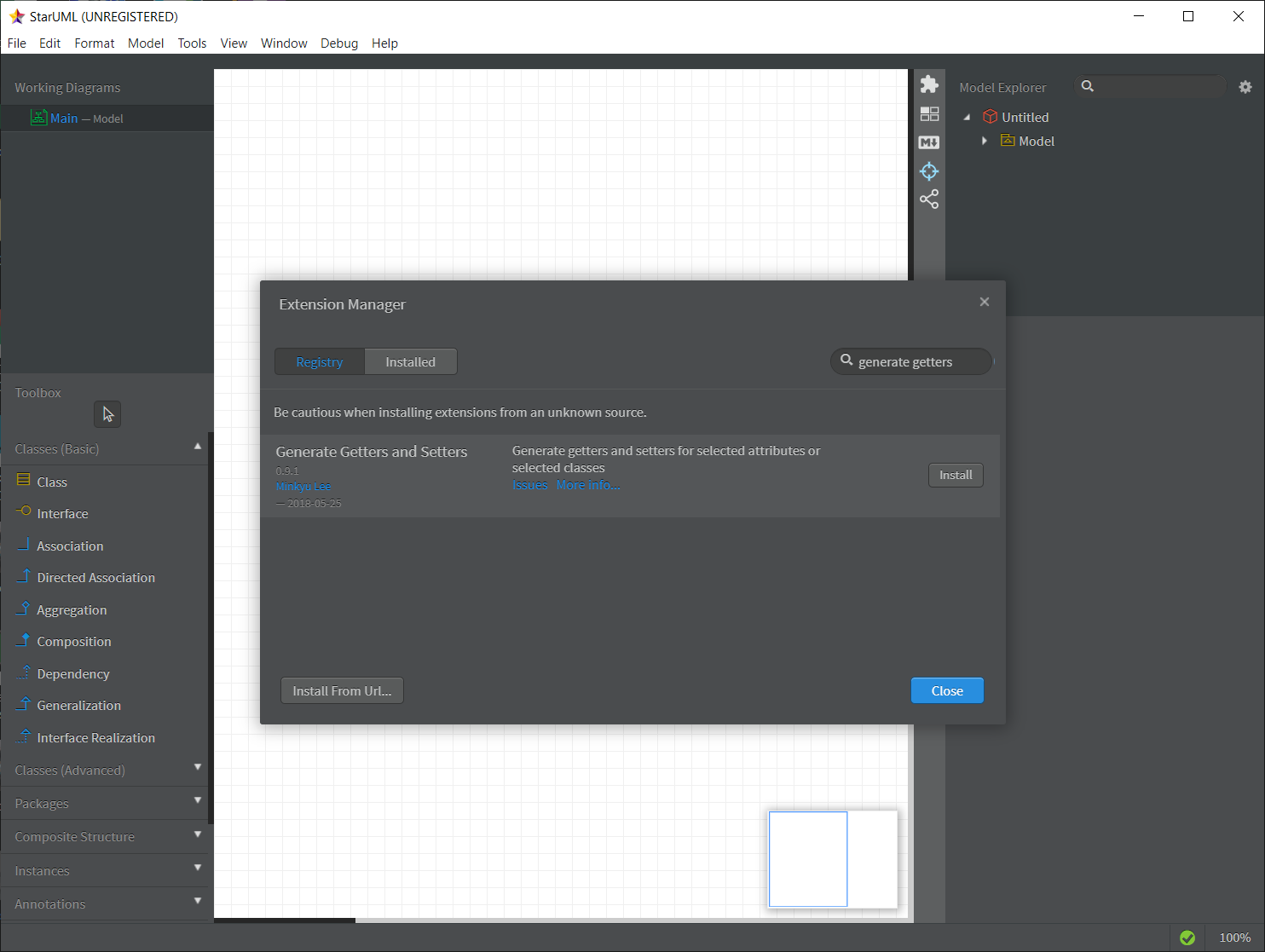
The source is (probably not needed): <https://github.com/staruml/staruml-java>

1. The Extension Manager should still be open. Search for, “merge generalizations”. Install this. It will say it needs to Reload. Choose: Close instead, as we will reload when we have installed all three.



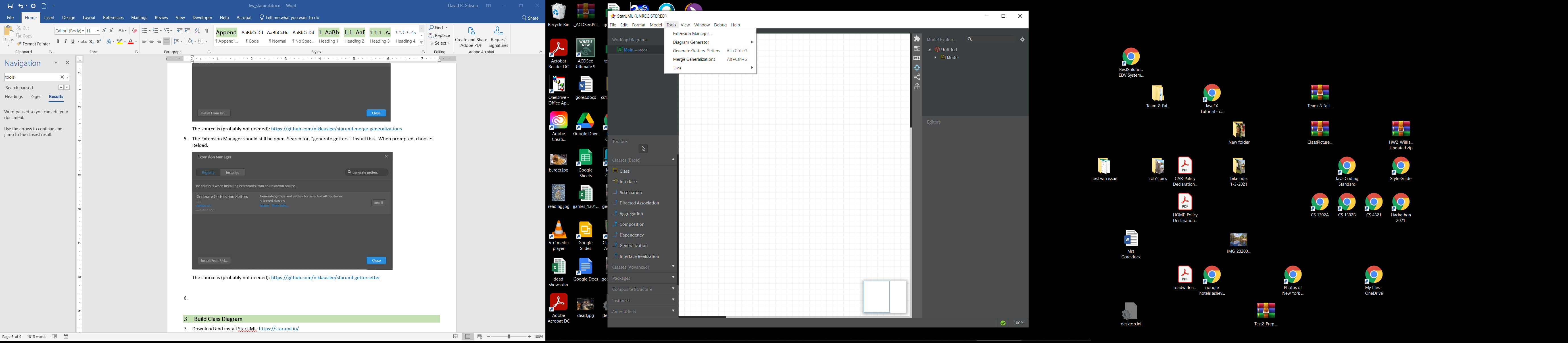
The source is (probably not needed): <https://github.com/niklauslee/staruml-merge-generalizations>

1. The Extension Manager should still be open. Search for, “generate getters”. Install this. When prompted, choose: Reload.



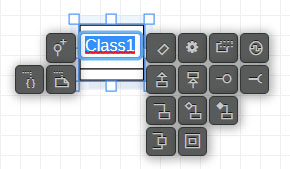
The source is (probably not needed): <https://github.com/niklauslee/staruml-gettersetter>

1. To verify the extensions have been added, choose the Tools menu and verify that you see the three extensions installed above.

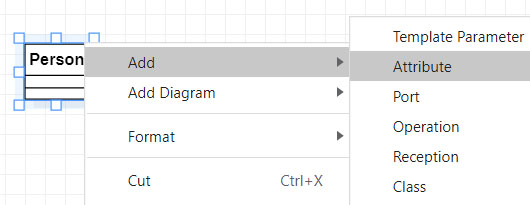


# Build Class Diagram

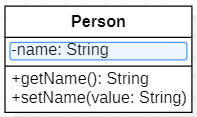
1. Choose: Model, Apply Profile, UML Standard Profile (v2) (not sure what this does, nor if we actually need it). It adds this node in the Model Explorer (upper-right of your screen).
2. In the Model Explorer (upper-right), right-click the *Model* node and choose: Add, Package. A *Package1* node is added to the Model Explorer (ME).
3. In the Properties dialog (lower-right), change the name from *Package1* to *animals.*
4. In the Model Explorer (upper-right), right-click the *Model* node and choose: Add Diagram, Class Diagram. Notice that a *ClassDiagram1* element was placed inside the *Model* node in the Model Explorer.
5. In the Toolbox (lower-left), select the Class icon, then click on the design surface. A class rectangle will appear. As shown below, with the class name selected, a number of contextual icons appear as shown in the figure below. We will be using some of these as we go along.



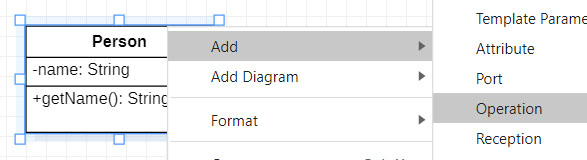
1. Change the name of the class to: *Person*. Then, Enter. (or, you could have done that through the Properties dialog (lower-right).
2. Right-click the class and choose: Add, Attribute.



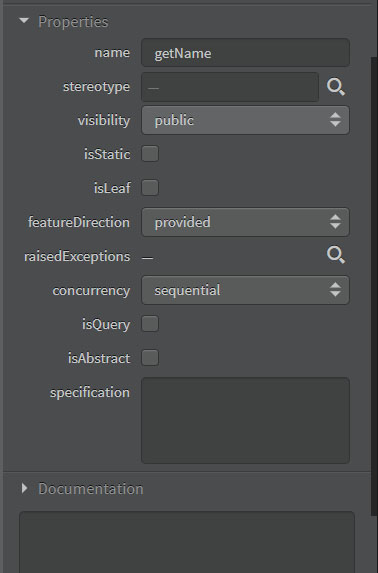
1. Double-click the text, *+Attribute1,* and type: -name:String and Enter. Then, click on the design surface and the rectangle will resize. Notice the *name* node in the ME under the *Person* node.
2. Select the *name* attribute in the diagram and then choose: Tools, Generate Getters Setters.



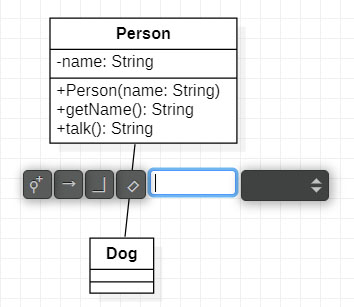
1. In the Model Explorer, right-click the setter (*i.e. setName*) and choose: Delete from Model (or this could have been done from the design surface).

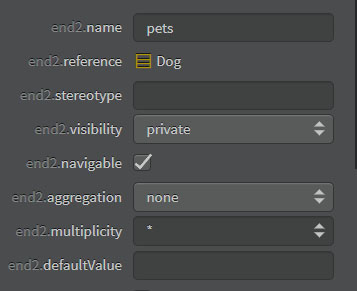


1. Right-click the class and choose: Add, Operation. *Operation1* was added in the diagram and as a node in the ME.
2. Double-click the text, *Operation1* and type: Person(name:String) and Enter. As we will see later, this did not create a constructor, it created a void method named Person. We will say more about this later.
3. In the Model Explorer, right-click the *Person* operation (not the class) and choose: Move Up.
4. Right-click the class and choose: Add, Operation.
5. Double-click the text and type: talk():String and Enter
6. Select the *getName* method (or any method or instance variable) and look at the Properties dialog (lower-right of UI). Just scroll through and see the options that are available. Notice that we can change the visibility, or make it abstract, *etc.*

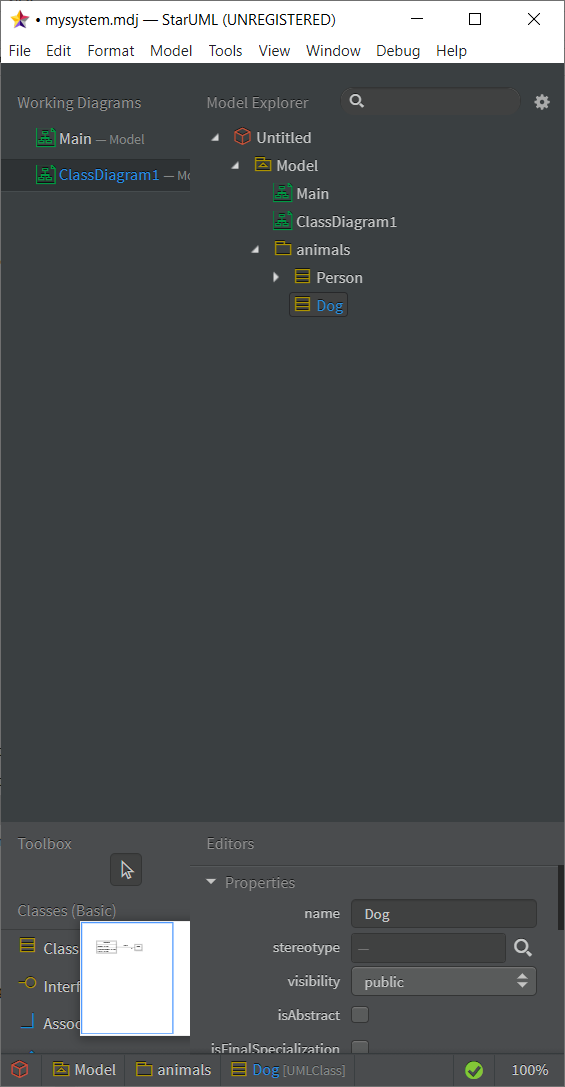


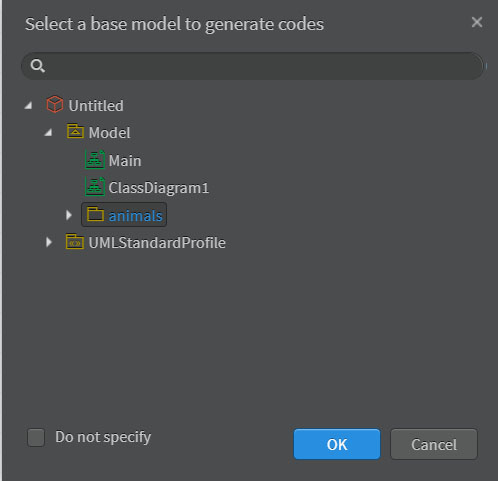
1. Add a class named Dog.
2. In the Toolbox (lower-left of UI), select the Association icon, then click, hold and drag on the Person class, dragging to the Dog class and release.



1. You are prompted to type a name. Don’t. Simply click on the design surface to dismiss the prompt.
2. Select the association and then look in the Properties (lower-right). Set the following properties:
3. end1.navigable – uncheck
4. end2.name – type: “pets”
5. end2.visibility – change to: private
6. end2.navigable – Make sure it is checked
7. end2.multiplicity – choose: “\*”
8. In Windows, use File Explorer to create a folder, *mysystem*, on your harddrive. This is where we will store your project and the code that is generated.
9. In StarUML, choose: File, Save, and navigate to the *mysystem* folder. Supply the project name: *mysystem.mdj* and choose: Save (you will get the nag screen again).

# Forward Engineer: Generate Code

1. To generate the code from these classes, they must be in a package. The classes shown in the Model Explorer are probably not in the *animal* package. If not, drag them there. The result is shown on the right.
2. Choose: Tools, Java, Generate Code.
3. Expand the Model node in the “Select a base model…” dialog. Then, select the *animals* package and OK.



1. A *Select Folder* dialog appears. Navigate to the *mysystem* folder and choose: Select Folder. Note the following:

* The code was generated, although no message appeared confirming this. We will inspect the code in just a minute.
* If you make more changes to your diagram and then repeat this process to generate code, it will NOT generate any code (and again, it will not tell you this). If you need to make changes to the diagram and you want to generate code again, it will only work if you either: (a) specify a new location, or (b) delete the package in File Explorer (Windows) first.

1. Use File Explorer (Windows) to navigate to the *mysystem* folder and there you will see the *animal* folder and inside the *Person* and *Dog* classes. Use Notepad (or any program) to open the *Person* class and note the following:

* It automatically provided a no-arg constructor.
* Our constructor has been turned into a method.

public void Person(String name) {

// TODO implement here

}

This can be corrected in StarUML with the proper syntax/property. However, I was unable to figure it out.

* For the 1-many relationship, it probably generated:

private Set<Dog> pets;

The default implementation in UML for a collection is *Set*. You can select the association and “check” the *end2.isOordered* property and then the code will generate a *List*. However, it puts a tag on the diagram that reads: “{ordered}” which I was unable to remove.

StarUML is template-driven which means that there are templates (probably XML) that specify how diagram elements are transformed into code. These can be modified as needed. However, I provide no instructions for this.

1. In summary, even though the code is not perfect, it is a starting place. And, as you’ll see next, when you start implementing the code and then reverse engineer it in StarUML, the diagram will be generated properly, without some of the caveats mentioned above.

# Reverse Engineer Code: Generate Diagram

1. Download the supplied code, *hw\_cd\_staruml\_code.zip* and unzip. There, you will find this path:

*com\yattasolutions\umllab\examples\shop*

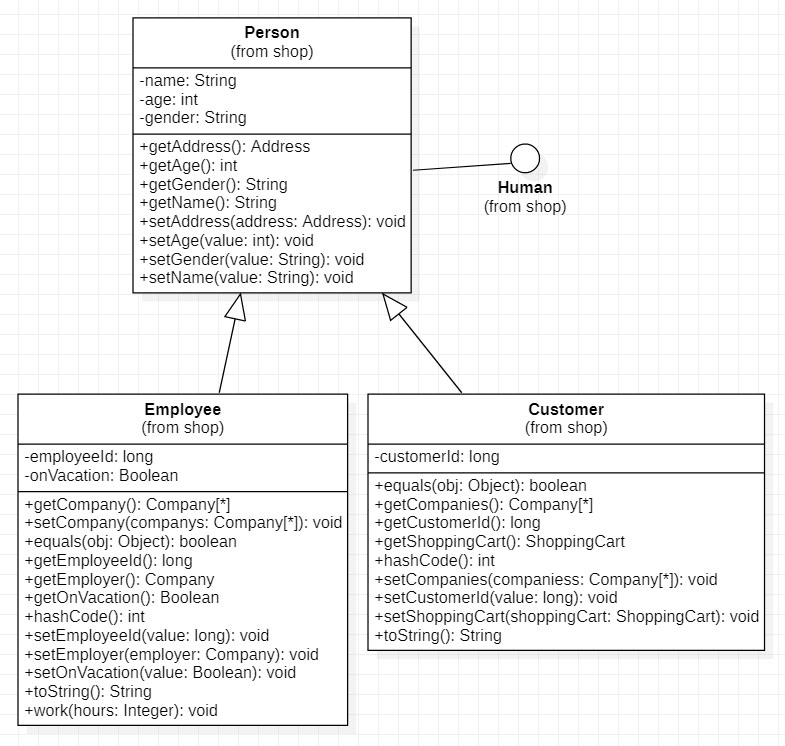
Inside, you will find a number of Java files. Next, we will reverse engineer this code to produce a class diagram.

1. Choose: Tools, Java, Reverse Code…
2. Navigate to the location of the top package: *com* (from the path above), select it, and then choose: Select Folder.
3. You will see a *JavaReverse* node in the Model Explorer with nodes for the package structure and all the classes.
4. Select the Model node in the Model Explorer, right-click, and choose: Add Diagram, Class Diagram.
5. From the Model Explorer, select the *Person* class (shown in figure below, on left) and drag unto the diagram (result shown in figure below, on right).

|  |  |
| --- | --- |
|  |  |

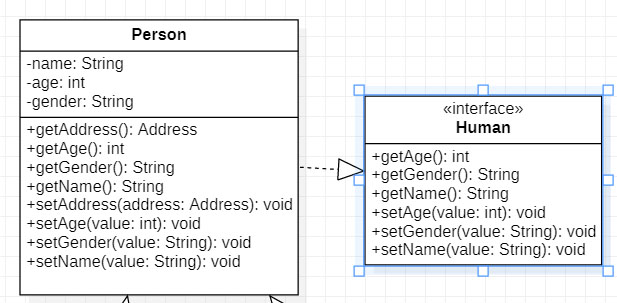
1. Repeat for the *Employee* and *Customer* classes, and the *Human* interface. Rearrange the diagram so that it looks as shown below (next page). Note:

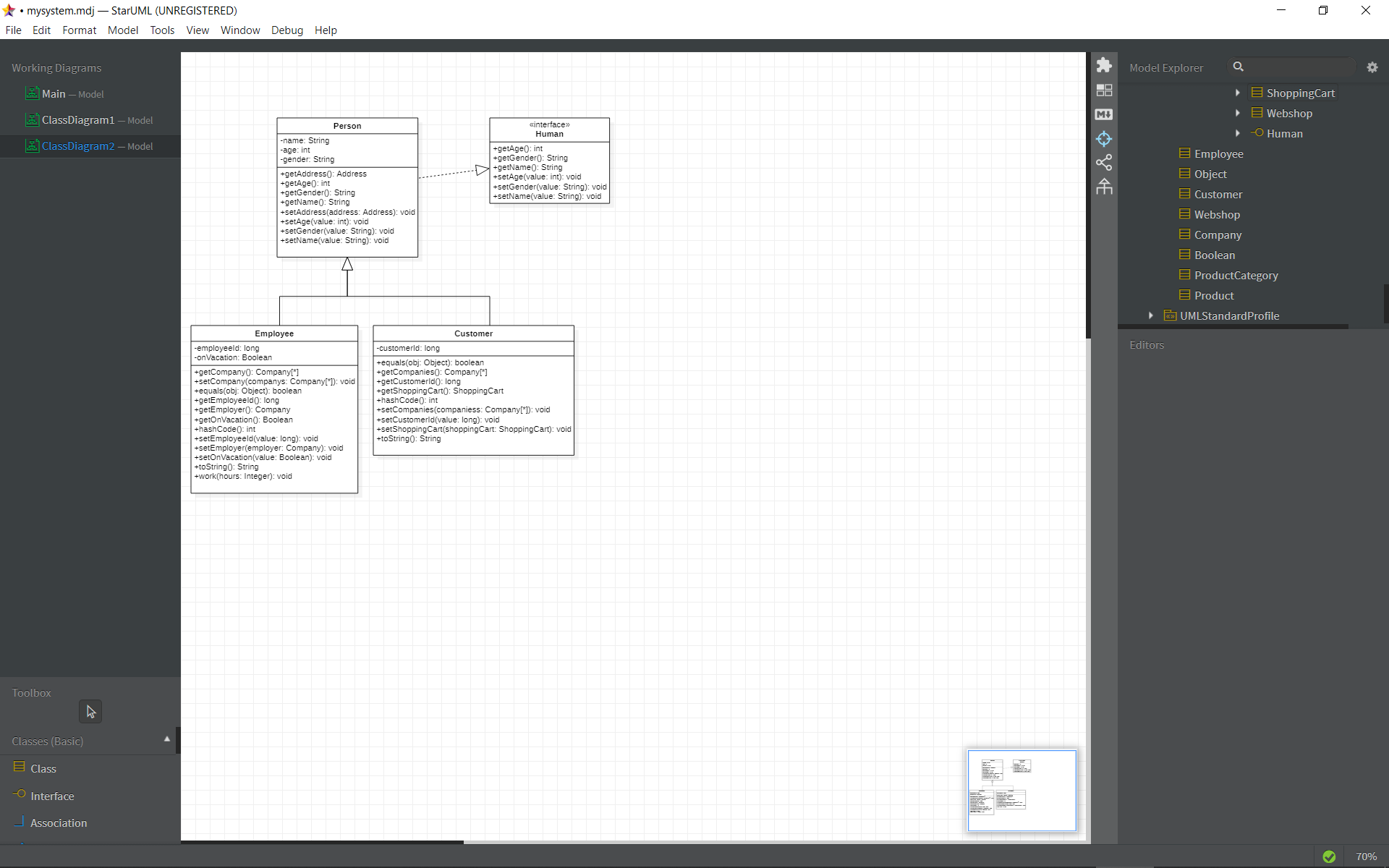
* These classes don’t have constructors. I borrowed the code, and that is the way it came. In the future, I may add these.
* Collections are indicated with the notation “[\*]”. For example, the *getCompany* method in the *Employee* class returns a collection of companies, denoted by: “Company[\*]”. In other words, it doesn’t specify the type of collection. Those of you who have had my CS 1302 course know that we would specify this as:
* getCompany():ArrayList<Company>
* The *getCompany* method is poorly named. It should be *getCompanies* since it is returning a collection of companies.



1. Next, we are going to:

* Hide the display of package (*shop*)
* Change the interface (*Human*) lollipop display to the standard rectangular one.
* Combine the two subclass arrows.

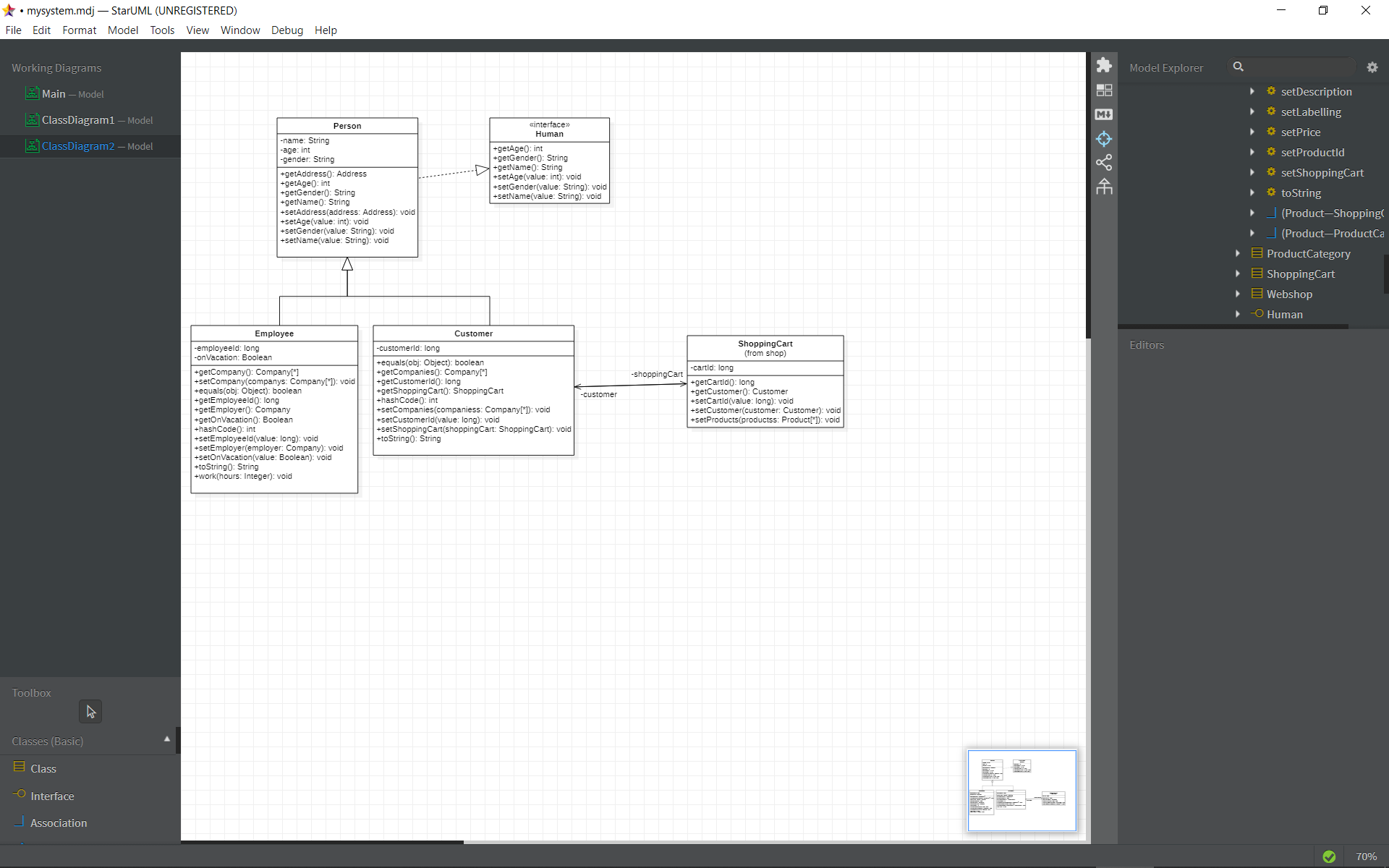
1. Select all diagram elements (the 3 classes and the interface) by dragging the mouse around them.
2. Right-click and choose: Format and then uncheck, “Show Namespace”
3. Select the *Human* interface, right-click, and choose: Format, Stereotype Display, Label.
4. With the *Human* interface selected, right-click, and choose: Format and uncheck, “Suppress Operations”
5. Select the superclass (*Person*) in the diagram and choose: Tools, Merge Generalizations. Note, you can select an endpoint of the resulting lines and drag them to a different location to make the diagram look nicer. The result should look similar to this:



1. In the box below, describe in a complete sentence(s) the relationship between the *Customer, Employee, Human, Person* classes and interface. If you are not sure what this means, see the Class Diagrams document on the Schedule in the Comments section beside the assignment.

|  |
| --- |
|  |

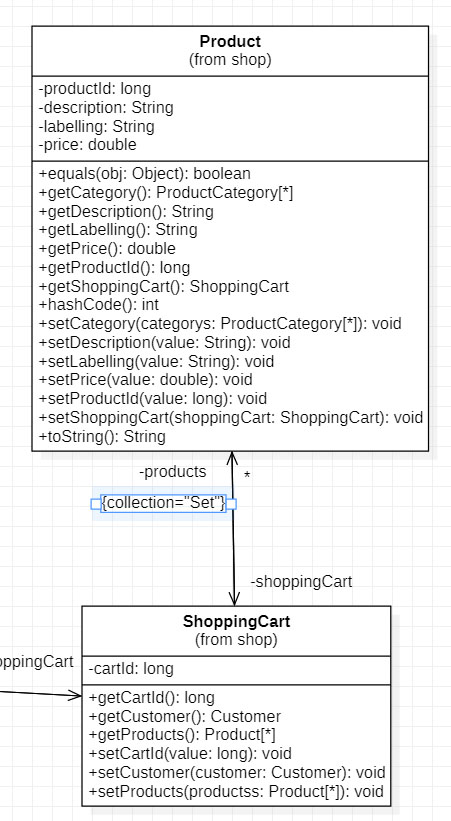
1. From the Model Explorer, select the *ShoppingCart* class and drag unto the diagram. Select a role name (*shoppingCart*) and drag to move so that it doesn’t intersect with the class. Do the same for the other role name.



1. In the box below, describe in a complete sentence(s) the relationship between the *ShoppingCart* and *Customer* classes.

|  |
| --- |
|  |

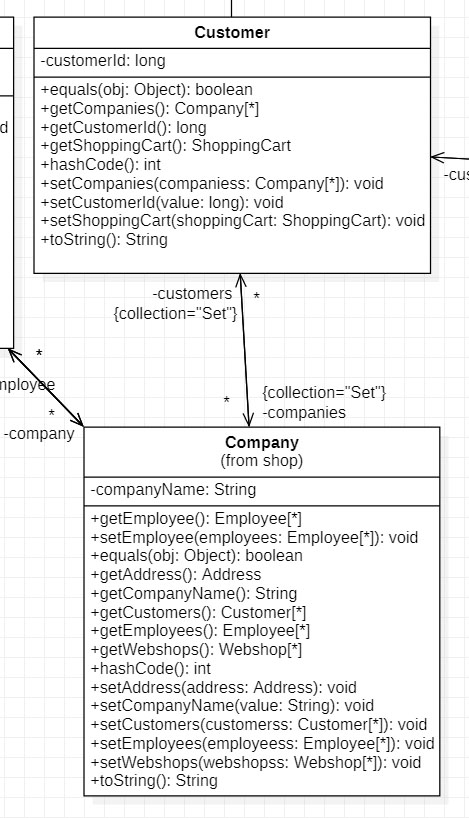
1. Drag the *Product* class onto the diagram. Rearrange to make it look nice and readable. I’m not sure how to hide the ‘collection=”Set” ’ comment, it is tied to the association. The default 1-many association in UML is *Set*. Probably with the aforementioned templates this can be changed.



1. In the box below, describe in a complete sentence(s) the relationship between the *ShoppingCart* and *Product* classes.

|  |
| --- |
|  |

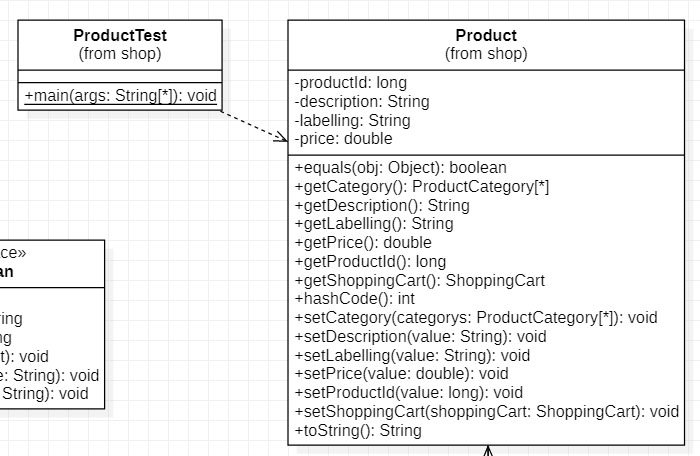
1. Drag the *Company* class onto the diagram. Rearrange to look nice. Don’t worry about the relationship between *Company* and *Employe*, just *Company* and *Customer.*



1. In the box below, describe in complete sentences the relationship between the *Company* and *Customer* classes. You do not need to describe the relationship between the *Company* and *Employee* classes.

|  |
| --- |
|  |

1. Try this: drag the *ProductTest* class onto the diagram. There will probably not be any relationship shown between the *ProductTest* and *Product* classes. If not, select Dependency icon from the Toolbox, click on *ProductTest* and drag and release on *Product*. You will be prompted to type something, just leave it blank.



1. In the box below, describe in complete sentences the relationship between the *ProductTest* and *Product* classes. (I’ll do this one for you!)

|  |
| --- |
| There is a dependency relationship between the *ProductTest* and  classes. A dependency relationship exists when a class uses another class, but does not possess it as an instance variable. We usually do not show dependency relationships on a class diagram unless there is something special we want to illustrate. |

1. Choose: File, Save.
2. Save your diagram: File, Export Diagram As, JPG (don’t use PNG, it is poor quality). Include the diagram below:

|  |
| --- |
| **To Conclude:**   1. **Save this document.** 2. **Save this document with a different name, perhaps: *hw\_cd2.docx.*** 3. **Delete EVERYTHING in this document except Questions (and your answers) 14, 16, 18, 20, and 24 (the diagram). Please do not delete the actual questions. The questions will be renumbered, probably 1,2,3,4, which is OK.** 4. **Turn in in the HW-CD-StarUML dropbox on Blazeview.**   **YOU ARE NOW DONE.** |

Appendix

1. Configuration

This post suggests that it is possible to configure StarUML, for example to always use text for the visibility of class members as opposed to icons. I just quickly read it and don’t see immediately how to do this. There are at least a handful of settings I’d like to make the defaults. I’ve searched a fair amount and this is the only link that seems related.

<https://sourceforge.net/p/staruml/discussion/510443/thread/8d2b9dfe/>

1. Resources & Sources

<https://www.clear.rice.edu/comp201/07-spring/info/staruml/>