

# UML Class Diagrams and Related Tools

By Roman Parise

## Basic Class Diagrams

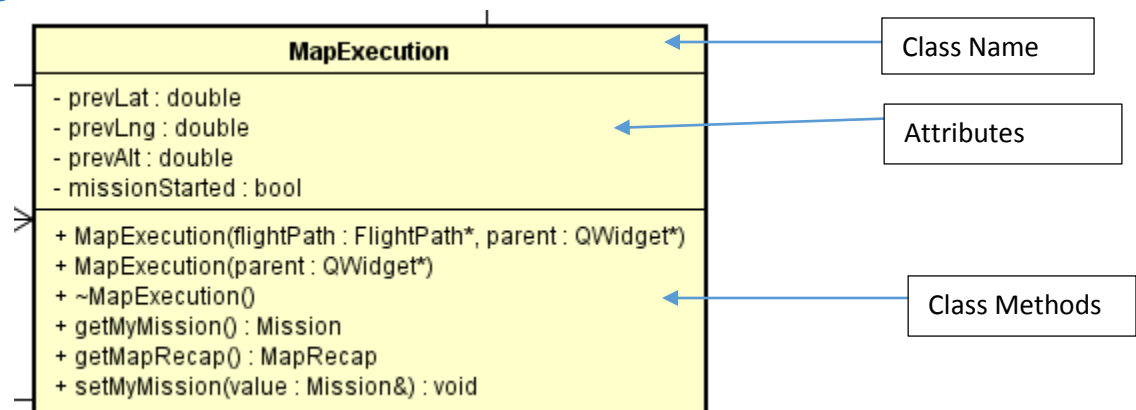


Figure 1

Figure 1 shows how a class is displayed in a UML diagram. The name of the class being documented goes at the top of the box. Attributes, or essentially class variables, go in the middle of the box. Attributes are documented with the following convention:

( visibility symbol ) nameOfAttribute : attributeType

“Visibility symbol” usually stands for one of the following symbols:

+ - public variable; can be seen by other objects

# - protected variable; can be seen by the class and classes that inherit this variable

-- private variable; can only be seen by its specific class instance and nothing else

Look at the first entry as an example: “- prevLat: double”. Because it begins with “-”, we know this is a private class variable. “prevLat” is the name of the variable. “double” indicates this variable is a double type. This corresponds to the following C++ code (in the header file):

```
private:
```

```
/* Other private members */
```

```
double prevLat ;
```

The bottom section corresponds to class methods, which includes functions, constructors, and destructors. Entries in this section take on the same form as those in the attributes section, except the attribute type is replaced with the return type of the method. Constructors and destructors exclude this return type.

Look at the fourth entry: “+ getMyMission() : Mission”. This is a public class method named “getMyMission” that returns a Mission type. In a C++ header file, this would be written as:

```
public:
```

```
/* Other public members */
```

```
Mission getMyMission() ;
```

Diagrams can also specify input parameters to class methods. Look at the second entry: “+ MapExecution(parent : QWidget\*)”. This is the constructor for the MapExecution class. Its input parameter is named “parent” and is of type “QWidget”.

## Inheritance and Relationships Between Classes

### Association

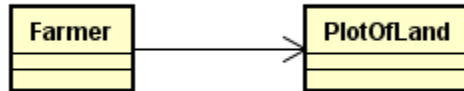


Figure 2

Figure 2 demonstrates a basic example of association. Here, the class, Farmer, is associated with the class PlotOfLand. This essentially means that Farmer has an instance of PlotOfLand. Notice that the arrow points from Farmer to PlotOfLand, not the other way around. This really means that whether Farmer is “navigable” from PlotOfLand is ambiguous. Essentially, Farmer has information about PlotOfLand, but it is not clear whether PlotOfLand has information about farmer. This is what is meant by navigability.

However, Astah code generation (discussed later) treats PlotOfLand as if it does not have an instance of Farmer. To be more explicit, use an ‘X’ on Farmer’s side of the arrow to show that the relationship is not true in the other direction.



Figure 3

Figure 2 corresponds to the following code in a C++ header file:

Farmer.h

```
class Farmer
{
    private:
        PlotOfLand plotOfLand;
};
```

However, this association is not very specific. Figure 3 demonstrates how to specify the instance name as well as its visibility.

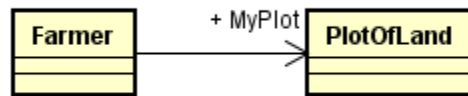


Figure 4

Farmer's PlotOfLand instance is now a public variable named MyPlot. The header file code would now look like this:

Farmer.h

```

class Farmer
{
    public:
        PlotOfLand MyPlot;
};
  
```

## Inheritance

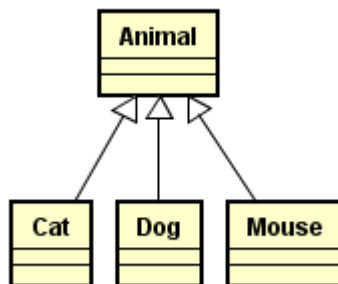


Figure 5

Representing class inheritance in UML diagrams is relatively straightforward. Figure 4 shows a superclass, Animal, and various subclasses, such as Cat, Dog, and Mouse. The arrow with the white triangular head represents inheritance. This diagram corresponds to the following C++ header file code:

Cat.h

```

class Cat : public Animal
{
};
  
```

# Using Astah with Doxygen

## Introduction

Astah is a program that can be used to generate diagrams for software projects, including UML and finite state machine diagrams. Doxygen is a tool for automatically generating documentation for a software project. These two programs can be used to both generate code from UML diagrams and reverse engineer C++ code to produce UML diagrams.

## Setup

Go to [astah.net](http://astah.net) to download Astah. Students are able to receive a free software license. Doxygen is freely available for download at [doxygen.org](http://doxygen.org).

In addition to the tools above, Astah requires the C++ Reverse Engineering Plug-In, so that it can generate UML diagrams from XML files generated by Doxygen. Visit <http://astah.net/features/cpp-reverse-plugin> to download the plug-in.

## Making UML Diagrams

Run Astah. At the start, select “Class Diagram” under “Create Diagrams”.

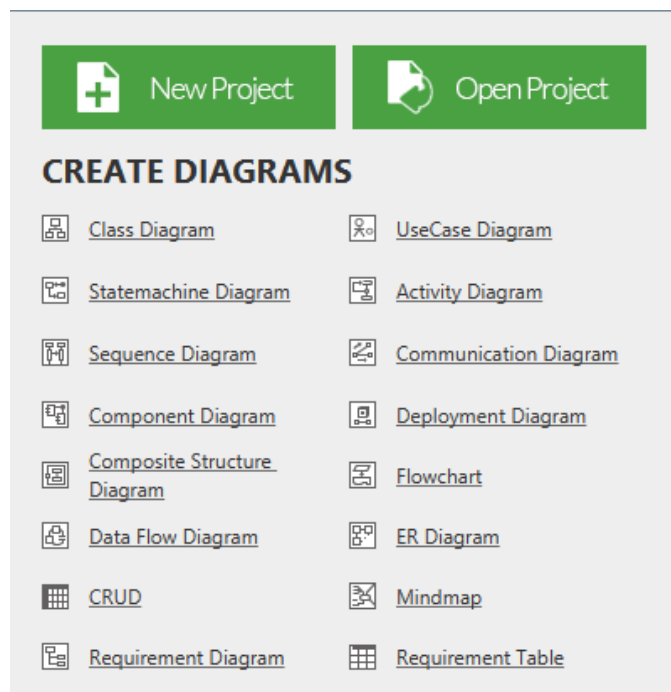



Figure 6

To make a class, select the  icon from the toolbar at the top. Then click on the class diagram to create the class.

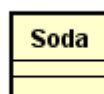



Figure 7

To make one class inherit another, select the  icon from the toolbar. Then click on the subclass, hold the click, and drag the cursor to the superclass before releasing.

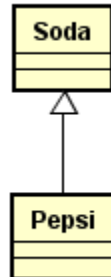



Figure 8

To establish an association between class, click the  icon. Click on the class that will possess the instance first and drag the cursor to the class of the owned instance before releasing.

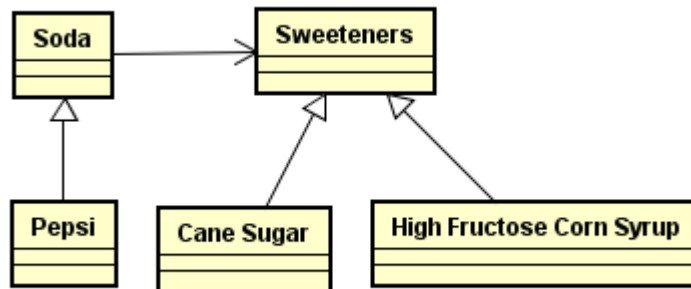


Figure 9

To name the instance and change its visibility, click on the association arrow, and navigate to the “Association End B” tab in the bottom left hand corner of the Astah interface. These options are available in this menu.

Constraint B	TaggedValue
Constraint A	Association End B
Constraint	Association End A
Base	Stereotype
Target	Class 1
Type Modifier	
Name	
Navigation	navigable
Aggregation	none
Initial Value	
Visibility	private
Close	

Figure 10

## UML Code Generation from Diagram

Consider the Cat, Dog, and Mouse example from Figure 4. To generate the .h and .cpp files, navigate to “Tools” -> “C++” -> “Export C++ ...”. See Figure 5 for a graphical view of this process.

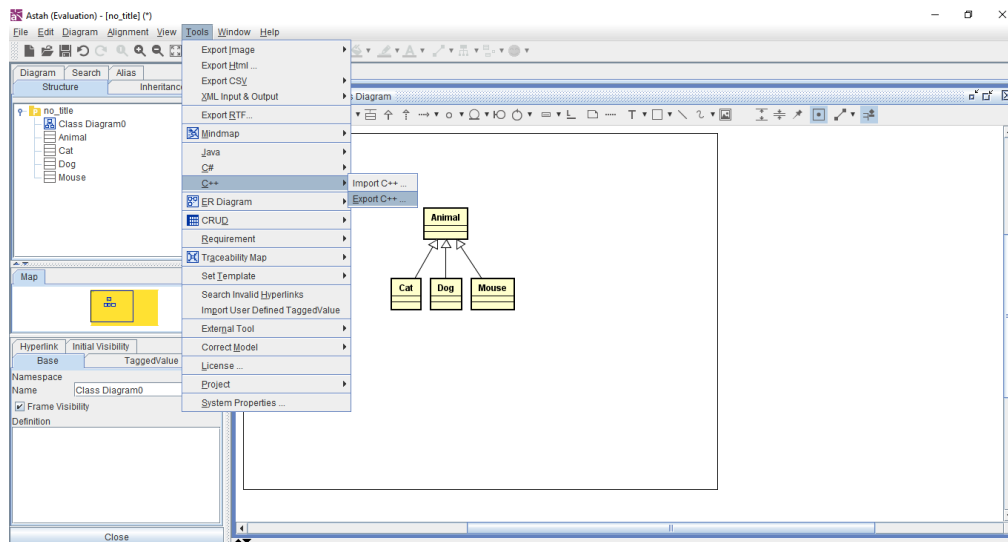


Figure 11

After selecting the “Export C++ ...” option, the user is now prompted to select a directory to which the files will be saved. Click “Select” when the desired directory is chosen.

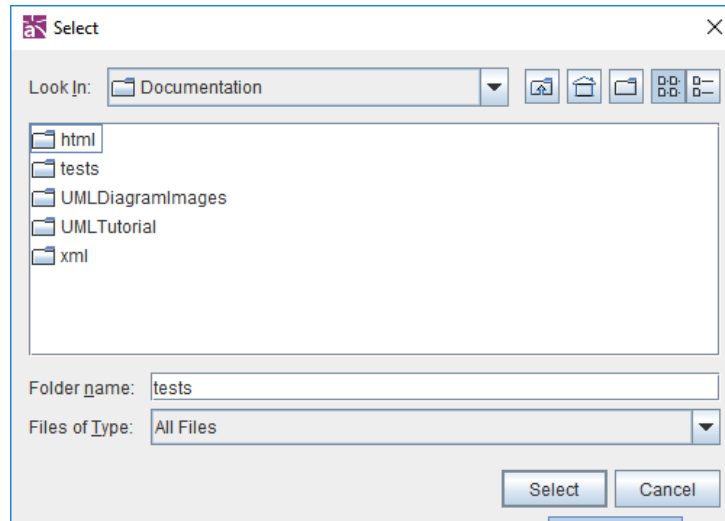


Figure 12

The user is now taken to the “Select Diagram Element” window. Select the diagram in the leftmost box. The candidate list will then populate with the various classes defined in the user’s diagram. These can be selected one by one by clicking the class name and then the darkened arrow. All of the classes can be selected by clicking the “>>” button.

After selecting the desired classes for which to generate code, the selected list is now populated. Click “OK” to generate the code.

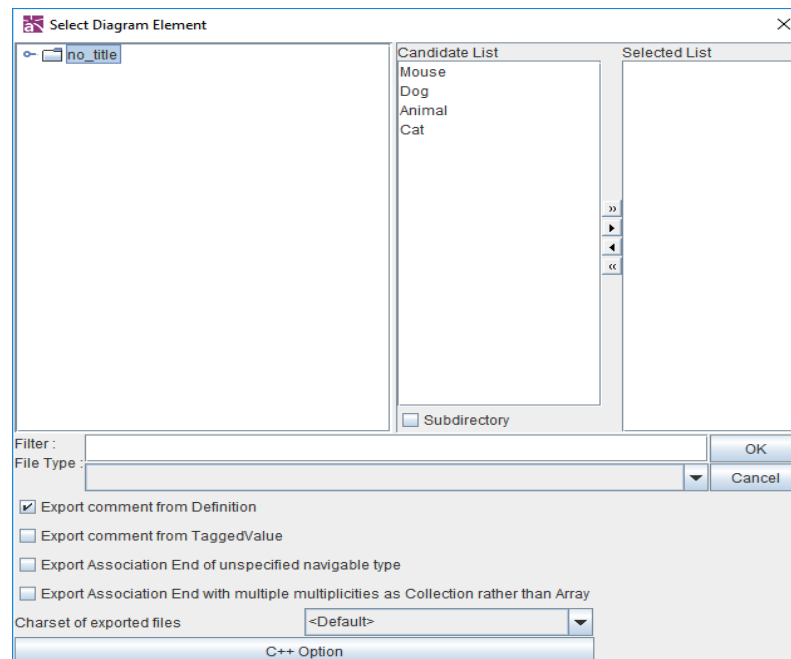


Figure 13

The selected directory should now be populated with the .cpp and .h files.



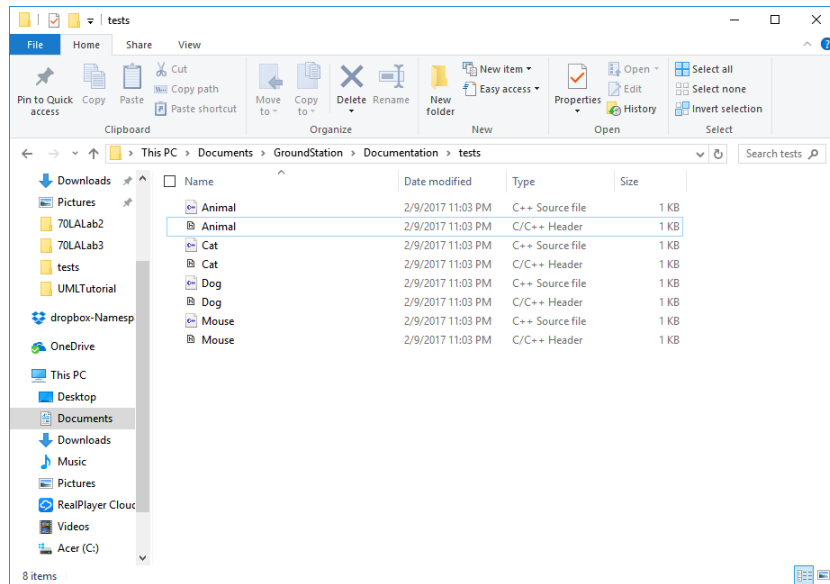


Figure 14

## UML Diagram Generation from Code

Classes Mechanic and Wrench, defined in respective header files, are used for this demonstration. Their source code is given below:

### Mechanic.h

```
#include "Wrench.h"

class Mechanic {
public:
    Mechanic();
    ~Mechanic();

private:
    Wrench MyWrench;
};
```

### Wrench.h

```
class Wrench {
public:
    Wrench();
    ~Wrench();
    bool isWrenchInUse();
};
```

```

        void useWrench() ;

private:

        bool wrenchInUse ;

};

```

First, Doxygen XML files will need to be generated. In Doxywizard, navigate to “Wizard” -> “Output” and select “XML”.

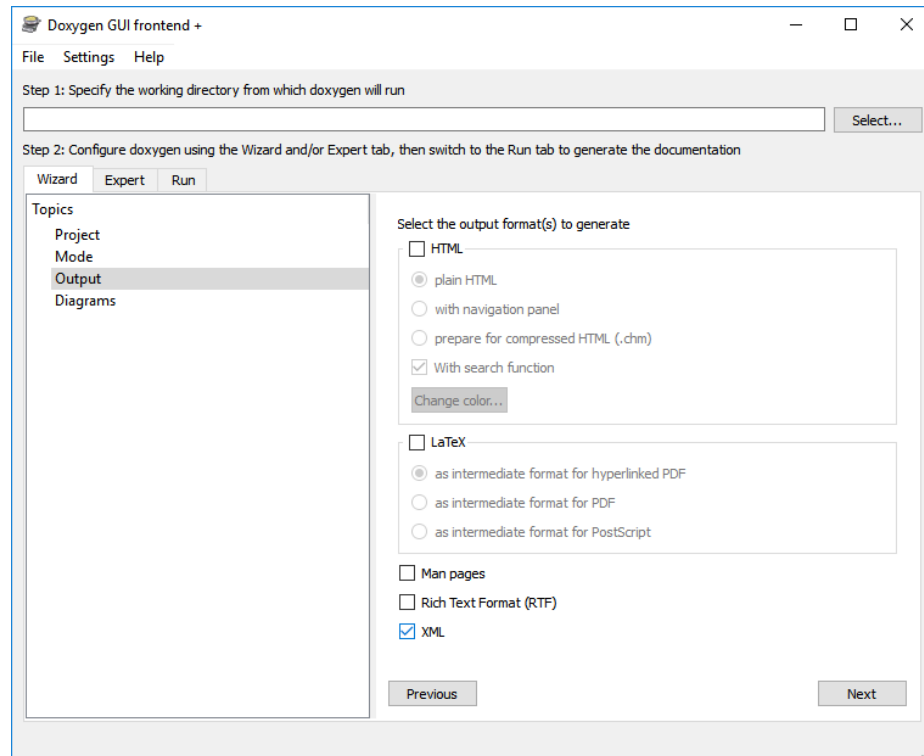


Figure 15

To parse code that does not contain Doxygen commenting, navigate to “Expert” -> “Build” and check “EXTRACT\_ALL”. This feature will be necessary for this example.

Run Doxygen. The XML files will be saved to the directory where documentation is generated under the subdirectory entitled “xml”.

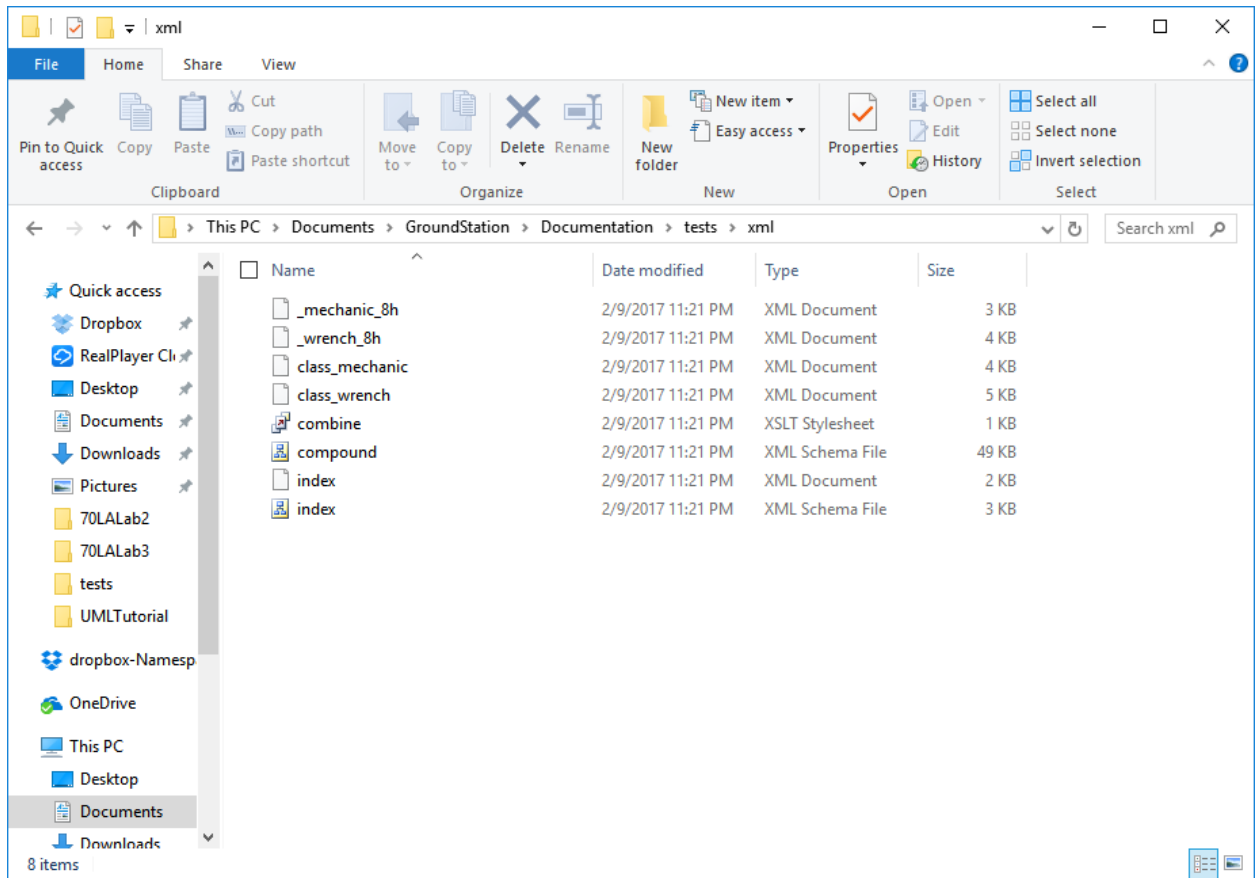


Figure 16

In Astah, navigate to “Tools” -> “C++” -> “Import C++ ...”. In the “C++ Reverse” window, select the directory of the Doxygen XML files for the project. Then click “Reverse”.

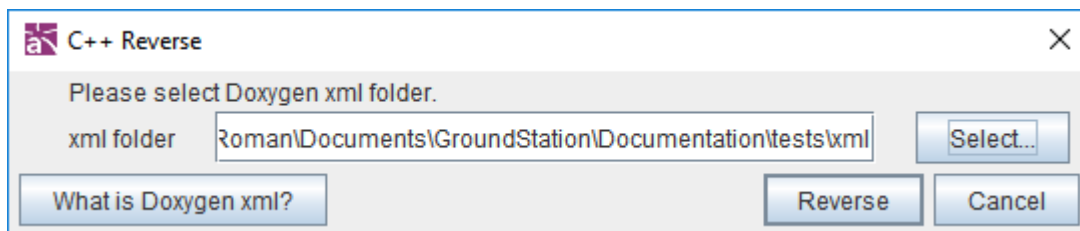


Figure 17

The classes from the reversed project should now appear under the “Structure” tab in the upper left hand corner of the Astah interface.

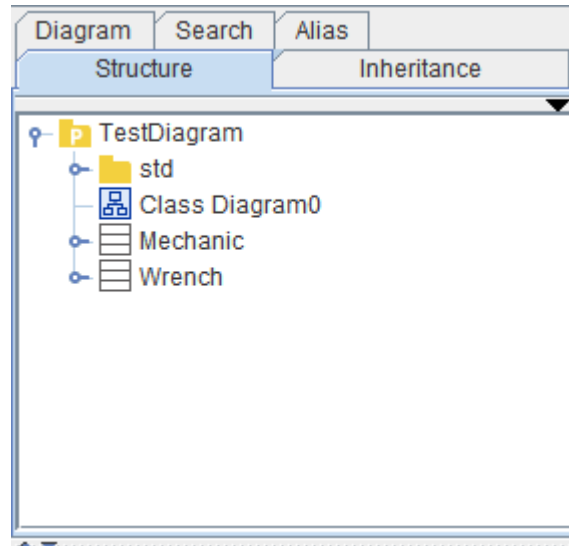


Figure 18

Right-click on one of the classes listed and select “Auto Create Class Diagram” or “Auto Create Detailed Class Diagram”. Figure 13 is an example of Mechanic’s generated class diagram.

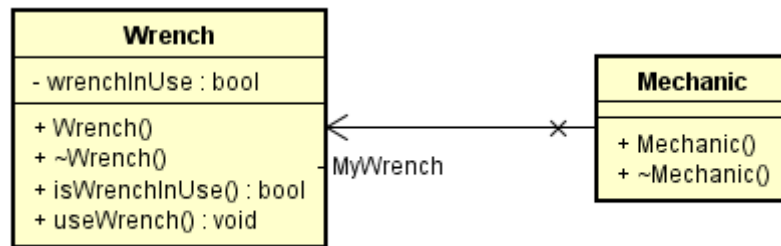


Figure 19