# CS225L Lab 7: UML Diagrams

# Learning Outcomes

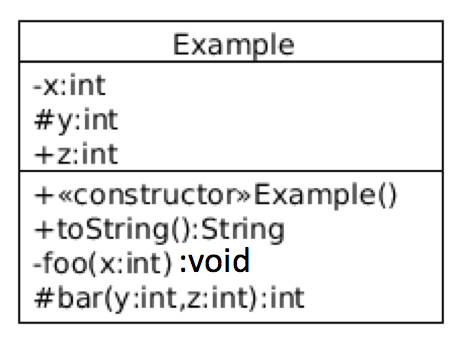
* What is UML
* How to create a UML diagram

# Pre-lab: Intro to UML Diagrams

**Overview**

Unified Modeling Language (UML) is a way of visualizing a software program using a collection of diagrams. The notation has evolved to be used for object-oriented design, but it has since been extended to cover a wider variety of software engineering projects. Today, UML is accepted by the Object Management Group (OMG) as the standard for modeling software development.

(ref: https://www.smartdraw.com/uml-diagram/)

**A Single Class Example**

Say we have the below class:

public class Example {

private int x;

protected int y;

public int z;

public Example() { ... }

public String toString() { ... }

private void foo(int x) { ... }

protected int bar(int y, int z) { ... }

}

*Rules:*

|  |  |
| --- | --- |
| private | - |
| public | + |
| protected | # |
| package-private (default) | ~ |
| static | \_\_\_\_\_\_ (underline) |

fields: <visibility> <field name> <data type>

methods: <visibility> <method name> <parameter type> <return type>

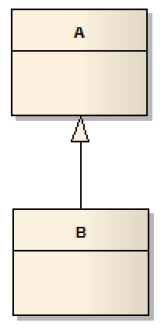
**Class Relationships**

There are four basic categories of class relationships:

1. Inheritance
2. Dependency
3. Association
4. Aggregation

*Inheritance - Generalization:*

public class B extends A { …… } public interface B extends A, C { …… }

*Inheritance - Realization:*

public class B implements A, C { …… }



*Dependency:*

The Dependency relationship is a simple one used to denote that a class relies (temporarily) on another class in its implementation.



Here the **BankAccount**'s **computeInterest**() method might use the **Math.pow()** method.

*Association:*

Association denotes a general relationship between classes which may change between consecutive executions of an application.

Example 1:

public class House {

// attributes (such as address) and code for class House go here...

}

public class Person {

private House home; // Person includes a reference to an instance of House

}

// other attributes (such as name) and code for class Person go here, which may make use of the 'home' reference }



Example 2:

public class House {

// attributes (such as address) and code for class House go here...

}

public class Person {

private List<House> homes; // Person includes a reference to an collection of Houses }

}



*Aggregation:*

Aggregation relationships are used to indicate that instances of one class *contain* or *are composed of* instances of other classes.

public class Course {

private List<Student> students; // reference to a collection of Students,

private Professor instructor; // a reference to an single instance of a Professor,

private Schedule sched; // and of a single Schedule

// other attributes and code for the Course class go here...

}



If the diamond is left empty, it signifies it is an aggregation. This relation is stronger than a simple association. In this case a Customer aggregates Orders.

If the diamond is black, this means it is a composition, which is even stronger than an aggregation because the aggregated class cannot be aggregated by other classes. Its "life" depends on the container.

(ref: https://stackoverflow.com/questions/15141678/what-does-a-diamond-sign-signify-in-uml-class-diagrams)

**Reading Materials**

1. From Ball State University: <http://www.cs.bsu.edu/homepages/pvg/misc/uml/>
2. From tutorialspoint: <http://www.tutorialspoint.com/uml/>

**Lab Activities**

**Introduction**

In this lab, you will begin to use UML class diagrams to visualize how classes are interconnected. The goal of this lab is simple: Create a UML class diagram by hand.

**Reverse-Engineering to UML**

1. Start by importing the lab files into Eclipse as an existing project.
2. From there, your task is to create a UML class diagram based on this given code. It needs to have the following qualities:

* The diagram includes all of the classes in the provided code. (You do not need to include classes from the Java API.)
* All of these classes are presented with all of their respective fields and methods.
* All of the fields and methods have type information.
* All of the fields and methods are appropriately annotated to show their code modifiers, e.g. public, static, etc.
* All of the relationships between the classes are noted and correct.
* All relationships have the correct multiplicity notation.

In short, “fully decorate” your class diagram.

**Create ONE diagram with all of these qualities** on one side of a sheet of letter-sized paper. You should include all 6 classes in the diagram, each class stands for **5** points ( in all). I strongly recommend using pencil for this lab.

**Post Lab Deliverables**

Your submission is your class diagram, plus your answers to the below questions (**10** points):

1. What information does a UML class diagram not provide to a viewer?
2. Provide feedback:

(a) What is your lab section? (Section # or day of the week)

(b) What did you like about the lab? What did you dislike?

(c) What would you change about this lab to improve it?

To be clear, you should turn in your class diagram and your post lab questions on a sheet of paper during the lab section.

Total Points from Lab 7: **40 Points**