University of Calgary
Department of Electrical and Computer Engineering
ENSF 614
Lab 4 – Fall 2024

M. Moussavi, Ph.D. P.Eng

This is a Group Assignment:

In this lab, you can work with a partner. Only groups of two are allowed. If you are working with a partner, you must only submit one copy of the lab report with both names. Please do not submit two copies.
Working with a partner means you should work together, not divide the work between group members. Otherwise, you will not learn about all the concepts or problems that each exercise aims for.

Objective:

Before we move to the next chapter of the course, which is design patterns and then application-level design techniques, you need to have a good understanding of the details of UML 2.5 class diagram. Therefore, the main objective of this assignment is to provide you with a few exercises on UML class notations.

Marking Scheme: (28 marks total)

- Exercise A: 4 marks
- Exercise B: 6 marks
- Exercise C: 3 marks
- Exercise D: 15 marks
- Exercise E: (no marks)

Important Note: Exercise E will not be marked nothing should be submitted for this exercise. However, exercises that will not be marked are as important as other exericises, as similar questions may appear in upcoming exams.

Method of Submission and Due Dates: You should submit your lab report (in PDF format), and associated files into the **Dropbox** created for lab-4 on the D2L.

Due Dates: Wed October 9, 2024, before 3:00 PM

Exercise A – UML Class Notation (4 marks)

Consider the definition of the following classes in **Java** and **C++:**

```
class Point {
    private: double x, double y;
    public:
    Point (double a, double b): x(a), y(b)
    {
    }
    double getx() const {return x;}
    double gety()const {return y;}
};
```

```
class Node< D >implements Cloneable
{
   Integer keyM;
   D itemM;
   Node <D>nextM;
   public Node(){
        keyM = itmeM = nextM = null;
   }

   public Node(D itemA, Integer keyA, Node < D> nextA)
   {
               itemM= itemA ;
               keyM = keyA;
               nextM = nextA;
   }
}
```

```
class Shape{
public:
  Shape(double x_origin, double y_origin, string name);
  ~Shape();
  Shape(const Shape& source);
  Point getOrigin();
  string getName();
  static double distance (Shape& the_shape,  Shape& other);
  virtual double area() = 0;
protected:
  Point origin;
  string shapeName;
};
```

Now use UML 2.5 class notations, and draw diagrams for each class, using StarUML that normally you should be able to download a trial version for free. You don't need to show the constructor destructor or assignment operator, getters and setters for any of the classes in this exercise.

What to Submit:

Submit your diagram as part of your lab-report in (PDF format).

Exercise B - Association and Multiplicity/Cardinality (6 marks)

What to Do:

Consider the following simple problem statements and draw the association relationships and the cardinalities among the classes expressed in each statement. You don't need to show attributes and operations in each class. The focus of the question is on multiplicity/cardinality. Here are the statements:

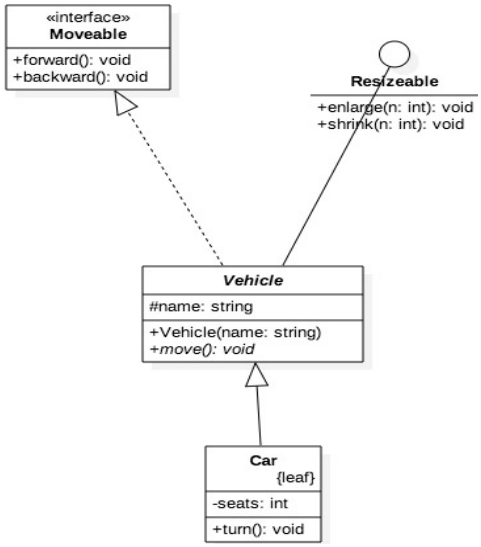
- The Department must use one or more Vector to store different type of data.
- Each Department must have many Professors, and one of the Professors assumes the role of Department head.
- Each Course must have one or more Sections and may have one or more pre-requisites., and each Section must have many Students.
- Each Professor must teach one or more Section of one or more Courses.

What to Submit:

Submit your diagram as part of your lab report in (PDF format).

Exercise C - From Diagram to Code (3 marks)

Assume the following class diagram belongs to a small application provided by a design engineer. Now your task is to convert the diagram to C++ classes.
Note: You don't have to worry about the implementation of any of the member functions or any logic behind this program; only write the definition of classes that normally goes into a header file.



Exercise D - Reverse Engineering a Java Program (15 marks)

What to Do:

In this exercise, your job is to reverse engineer a Java program. You should use StarUML and draw a class diagram for this program that is posted on the D2L, in a compressed file called ExD. Your diagram should include all details that can be extracted from given files, such as:

- Relationship among classes (association, aggregation or composition, inheritance, realization, etc.). For associations, make sure to use appropriate label and navigability.
- Multiplicity/cardinality
- Only important operations (constructor, clone method, getters and setters' methods are not needed)

- All attributes

Notes:

- Make sure to use correct notations for **attributes** and **behaviours,** as discussed during the lectures.
- Whenever applies, use appropriate **stereotypes**.

What to Submit:
Submit your diagram as part of your lab report in (PDF format).

Exercise E - Reverse Engineering a C++ Program (Not marked)

In this exercise, you will reverse engineer a C++ program. You should use StarUML and draw a class diagram for the program that is posted on the D2L in a compressed file called ExE. Your diagram should include all details that can be extracted from given files, such as:

- Relationship among classes (association, aggregation or composition, inheritance, realization, etc.). For associations, make sure to use appropriate label and navigability.
- Multiplicity or cardinality
- Only important operations (**ctor, copy ctor, assignment operator, getters and setters are not needed**)
- All attributes

Notes:

- Make sure to use correct notations for attributes and behaviours.
- Use appropriate stereotypes whenever needed.
- For information in the header files `xo_constants.h`, and `distribs.h` you can either ignore them, or use `UML interface` to show them.

What to Submit:
Nothing to submit.