# Assignment 7
# COMP 86

In this assignment we will add zooming plus your final touches to the simulation/game of the previous assignments, as well as creating a UML diagram.

## Final Touches

This is the culmination of your simulation/game, so it is also your chance to put any finishing touches on it or polish it up in any way you like, in addition to the specific requirements. As before, you need not be restricted by what you did in your previous assignments, but you should still provide all the features required by the previous assignments.

## Zooming

Provide commands to allow the user to zoom in or out of the map of the vehicles, to see a broader or narrower view. Your vehicles should continue to remember their "real" locations in the universe, but they will be displayed at different pixel positions on the screen, depending on the zoom setting. This requires adding another layer to your program, for translating between vehicle locations and screen locations; it will require scaling the locations (multiplying by a constant), and possibly also offsetting them (adding constants) to provide a reasonable center point for the zooming action. Don't forget that the zoom setting will also affect mouse picking, and it may have other implications in your program. You should not enlarge or shrink the vehicle icons/pictures themselves, just the positions of the vehicles on the screen (as would be typical in a real radar or other display).

## UML Diagram

For this assignment, also submit a UML diagram of your classes and the relationships among them. It should show the main classes in your system, inheritance relationships, and aggregation (containment) relationships -- and collaboration (uses) relationships if you can fit them in.

Don't worry about perfect UML syntax, since it varies. And if the diagram is too complex or too cluttered, you can omit some of the details.

Please submit your diagram in PDF if possible. It can be a PDF file from a drawing program or a scan or photo of a handwritten paper document from your phone camera.

## Program Design and Practices

*(The rest of this still applies from previous assignments)*

Your program design should exploit the features of object-oriented programming (encapsulation of code and data, support for abstract data types, polymorphism/overloading, inheritance).In particular, object-oriented programming provides us a good way to handle the various data needed in callback routines.

You should use objects to encapsulate each interactive widget with the routines and data you need to use it or pointers to other objects containing such.

You should provide an object for each interactive widget or small group of widgets you create. That object should hold anything you need to remember about the widget from one callback to another, all the data pertinent only to the command for that widget or that you need to operate this control, (including, in most cases, a pointer to the model or other outside object needed to perform the actual action the user requested), and the widget's own listener callback routines.

If you have several widgets that share some behavior or properties, you should organize your objects into an appropriate inheritance hierarchy.

You will have other data that must be accessed by several widgets, particularly shared information about the state of the program or global information about the state of the user interface. Provide additional classes and objects for holding this kind of information.

Remember to trigger your drawing to repaint itself explicitly whenever one of your commands causes a change that should be reflected on the screen. And remember that the way to change the screen is first to change the data stored your classes and then to trigger the repaint.

You should follow these general Java programming practices:

- Make all instance variables of your classes **protected** or **private**. If you need access outside of the class, provide it with *set* and/or *get* methods; don't access protected the variables from outside the class (even though Java -- unfortunately -- allows us to).
- Avoid most global variables or widely-accessible public variables; pass the data you need explicitly.
- Put each class in a separate .java file.

## Design Documentation

In addition to your program, submit documentation about the design of your system in these forms:

- An outline showing the inheritance hierarchy
- An outline showing the aggregation hierarchy (which objects contain or "own" which other objects)
- A list showing "uses" or collaboration relationships (which objects use which other objects to perform functions)
- The information hiding "secrets" of each of your classes (i.e., what design decisions are entirely encapsulated within that class).

Submit this documentation electronically in text form, along with brief instructions for how to compile and run your program. Include it as part of the `readme` file that you submit with your assignment.