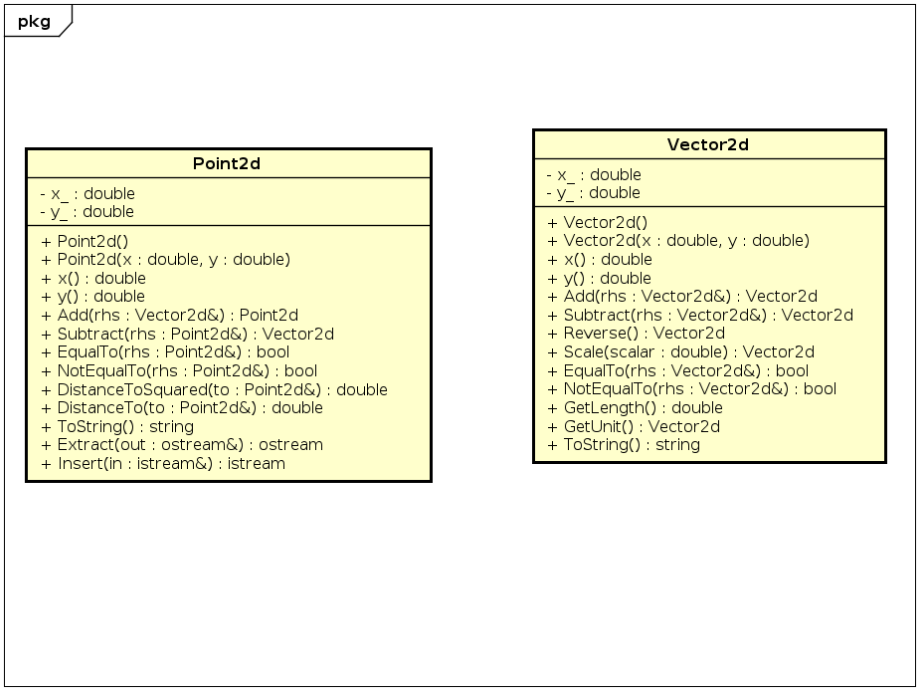
You shall upload a zipped (AND ONLY ZIPPED) archive to Blackboard containing the directory hw4 and:

1. A `hw4/vector2d.h` file with your includes and class declarations,
2. A `hw4/vector2d.cc` file with your class definitions,
3. A `hw4/point2d.h` file with your includes and class declarations, and
4. A `hw4/point2d.cc` file with your class definitions, at least.

We are yet again attempting a library. This time, it is a pair of classes—Point2d and Vector2d. These classes implement the behavior of two-dimensional points and vectors in a Cartesian coordinate system.

They are described in the UML below as well the attached header files.



In an effort to force you to read others' code, I have provided you with a fairly detailed set of unit test functions demonstrating the functionality. IJ and I will be referring any questions answered by reading the provided Unit Tests to the provided Unit Tests. Questions about the tests themselves or beyond the tests will, of course, be answered.

You may compile and link to them to ensure that your classes meet all requirements. The behavior demonstrated in the tests should well-define the expectations of the classes' operations.

The operations are further described as follows:

1. Vector2d—a two-tuple representing direction and magnitude in the Cartesian coordinate system.
   (a) Add and Subtract: returns the result of the operation where the calling object is the left-hand side of the operation
   (b) Reverse: returns a vector with the opposite direction of calling object
   (c) Scale: returns the result of scaling the calling object by the given scaler
   (d) Equivalence: true if approximately equal, false if not
   (e) GetLength: returns the calling object's length
   (f) GetUnit: returns a unit vector—a vector of magnitude 1.0 and the same direction as the calling object
   (g) ToString: returns a string representation of the calling object, notice that std::to_string from C++ string library is used to convert the floating point values
2. Point2d—a two-tuple representing location in the Cartesian coordinate system.
   (a) Subtract: returns the magnitude and direction from the calling object to the Point2d parameter as a Vector2d
   (b) Add: returns a Point2d offset from the calling object by the Vector2d parameter
   (c) Equivalence: true if approximately equal, false if not
   (d) DistanceTo and DistanceToSquared: return the Euclidean distance(s) from the calling object to the Point2d parameter
   (e) ToString: returns a string representation of the calling object, notice that std::to_string from C++ string library is used to convert the floating point values
   (f) Extract: extracts the values from the calling object as (x, y) and adds to the ostream parameter, then returns the ostream parameter
   (g) Insert: reads two floating point values from the istream and adds them to the two attributes of the calling object, then returns the istream parameter.

As in the previous assignments, you will receive up to 100% credit for turning it on the day it is due and will lose 25% per day late.

There are 10 points possible in the tests and another 3 possible in the styling. Note that the problem is only worth 10 points. That implies that there are up to 3 bonus points available in this assignment.

The End.