

# Assignment 0

Software Design Engr 5895, 2016

Due Jan 15 @ 11:30PM.

For each question you will be marked on programming style as well as correctness. To see my opinion about what constitutes good programming style see <http://www.engr.mun.ca/~theo/Courses/ds/pub/style.pdf>. In short:

- All .java files must be professionally commented; in particular, each file should contain a comment header that gives your name, student number, and mun email address. Each subroutine and class should have a comment at the start of it. I encourage you to use the “javadoc” conventions for comments.
- Code and comments must be consistently indented; tab stops should be set every 4 characters.
- Names must be chosen carefully and spelled correctly. (Use names starting with lower case letters for variables and methods; use names starting with upper case letters for classes and interfaces.)
- Use subroutines to avoid redundant coding.
- Keep control structures and data structures simple.

All classes must be tested by you prior to being submitted. You are welcome to share test code with each other.

The assignment is to be done alone. Each file should contain the following declaration in comments near the top. “This file was prepared by [your name here]. It was completed by me alone.”. If you obtained help in doing the assignment, do not include this declaration, but rather an explanation of the nature of any help that you received in doing the assignment.

## **Q0. Expressions**

You will implement a set of immutable classes representing expressions in  $x$ . Use at least one abstract class (with at least one abstract method). Your classes should be subtypes of the following interface `expr.Expression`.

```
interface Expression {  
    double value( double x );  
}
```

```
}
```

Your classes should be able to represent expressions such as  $\sin(2x + \pi/2)$ . Of course calling `value` on an object representing this expression with an argument of say 0.39270 would give a value of 0.70711.

In addition to implementing the `value(double)` method, override the `toString()` method, which is inherited from `java.lang.Object`, so that your `Expression` objects can be converted to readable strings.

You should also write a class for producing objects that represent expressions. This class should be called `expr.ExpressionFactory` and should declare and implement at least the following methods

```
public Expression add( Expression a, Expression b )
public Expression subtract( Expression a, Expression b )
public Expression multiply( Expression a, Expression b )
public Expression divide( Expression a, Expression b )
public Expression constant( double value )
public Expression x( )
public Expression sin( Expression a )
public Expression cos( Expression a )
public Expression tan( Expression a )
public Expression ln( Expression a )
public Expression exp( Expression a )
public Expression parenthesized( Expression a )
```

So our example expression can be constructed, printed, and evaluated via the following code

```
ExpressionFactory f = new ExpressionFactory();
Expression a = f.multiply( f.constant(2.0), f.x() );
Expression b = f.divide( f.constant( Math.PI), f.constant(2.0) );
Expression c = f.sin( f.add( a, b ) );
double x = 0.39270;
System.out.println( "The value of " +c+ " at " +x+ " is " +c.value(x) );
```

The library class `java.lang.Math` will be helpful for doing the calculation. Test your classes with the supplied JUnit tests.

### Q1. Graphing

Create a class `expr.ChartData` with a 0 argument constructor and following methods

```
public void setExpression( Expression a )
public Expression getExpression( )
public void setXRange( double xMin, double xMax )
public double getXMin( )
public double getXMax( )
public void setYRange( double yMin, double yMax )
```

```
public double getYMin( )  
public double getYMax( )
```

The following class invariants should be respected

```
getExpression() != null  
Double.NEGATIVE_INFINITY < getXMin( )  
getXMin( ) < getXMax( )  
getXMax( ) < Double.POSITIVE_INFINITY  
Double.NEGATIVE_INFINITY < getYMin( )  
getYMin( ) < getYMax( )  
getYMax( ) < Double.POSITIVE_INFINITY
```

The way you will use to ensure these invariants are respected is by using preconditions for the setter methods —i.e. we will shift responsibility from the class to its clients— and by using reasonable defaults in the constructor. These preconditions should be checked using `util.Assert`, which I will supply. The class invariants can be checked by calling a private invariant method at the end of the constructor and of each mutator. The invariant method uses `util.Assert` to check the invariant.

Once you have implemented this class you should be able to test it with the supplied JUnit tests and run the supplied.

Bonus. You might want to extend the graphing application to support more functions and operations. To do this, you will need to modify the parser and you will need to use JavaCC to regenerate the Java classes that make up the parser.

Submission: Zip (use zip format) up all files that you have modified or created and submit via D2L dropbox.

