You should use Antlr to generate code as detailed below. Please attempt all of the questions before the exercise class and bring your solutions along.

1. In the lecture notes we looked at the `ExtractInterface` example where we generate an interface declaration for any given Java class. Finish this example off by copying `import` statements from the class to the interface declaration and by using a TokenRewriter to make the base class implement the declared interface.

Now modify the example further by only listing public methods in the interface.

2. Use an ANTLR visitor implementation to generate Java code for a small language of Vector Algebra Transforms. We’ll only use vectors of 3 dimensions. The input model is a named transform that lists variables to be used as input vectors and an expression of Vector Algebra (with operations of vector addition, subtraction, scalar multiplication and cross product). I have prepared an ANTLR grammar for this:

```antlr
grammar VecAlg3d;

trans : 'transform' ID ':' inputs NEWLINE expr ;
inputs : ID ( ',' ID)* ;
expr returns [int nodeId ]
    : expr '+' expr  # VectorAdd
    | expr '-' expr  # VectorSub
    | expr 'x' expr  # CrossProduct
    | scalar expr    # ScalarProduct
    | ID             # UseInput
    | '(' expr ')'   # Parens
    | '(' INT INT INT ')' # Const
    ;

ID : [a-zA-Z]+ ;    // match identifiers
INT : [0-9]+ ;      // Define token INT as one or more digits
NEWLINE: '\r'? '\n' ; // Define newline
WS : [ \t\r\n]+ -> skip ; // Define whitespace rule, toss it out
```
So an example input model is

\texttt{transform Foo : A , B}  
\texttt{A + (5 B) }

The generated Java code should provide a class named \texttt{FooTransform} with a public static method named \texttt{doTransform} that accepts instances of the class \texttt{Vec3d} as parameters as indicated by the input model. The method \texttt{doTransform} should contain code that calculates the vector operation and returns the output \texttt{Vec3D}.

All the required files are available on the module webpage. Run the ANTLR tool on \texttt{VecAlg3d.g4} to generate the Lexer, Parser and Base Visitor code. Make sure you use the -visitor flag. I’ve also provided \texttt{GenerateTransform.java} to actually run the generator once you have written the \texttt{VecTransformVisitor.java} class.

Hints: think about what information needs to be passed up the parse tree as part of the traversal. I also found it useful to count the nodes that I had visited to give them each a unique ID. For this exercise, the Java code does not need to be concise, pretty or efficient. For example, I used a new \texttt{Vec3D} object at every operation node. The code generated by my solution for the above example input is

```java
public class FooTransform {
  public static Vec3d doTransform(Vec3d vA, Vec3d vB) {
    Vec3d vec0 = new Vec3d();
    Vec3d vec1 = new Vec3d();
    Vec3d vec2 = new Vec3d();
    Vec3d vec3 = new Vec3d();
    vec0.x = vA.x;
    vec0.y = vA.y;
    vec0.z = vA.z;
    vec1.x = vB.x;
    vec1.y = vB.y;
    vec1.z = vB.z;
    vec2.x = 5 * vec1.x;
    vec2.y = 5 * vec1.y;
    vec2.z = 5 * vec1.z;
    vec3.x = vec0.x + vec2.x;
    vec3.y = vec0.y + vec2.y;
    vec3.z = vec0.z + vec2.z;
    return (vec3);
  }
}
```