Usage:

Import -

- Netbeans
  The program is implemented in netbeans, so it should be as easy as opening a project and select this project. To start the program, click the green triangle thingie.
- Eclipse
  Here you have to make a new java project, in the project wizard, select import -> general -> file system -> next. Then select the base folder of this project in the form from directory, then click finish. To start the program, click the green triangle thingie.
- Git
  If none above works, or if you really enjoy git, then you can just clone the project from github using this URL: git://github.com/haavakno/INF3110_Mandatory1.git

Run:
When you run the program: Each test case given in the assignment text are evaluated and printed.

Implementation:

PrettyPrint:
Each class implements the interface Handler, which extends the interface IprettyPrint. IprettyPrint contains two methods

1. prettyPrint – if the object hasn't been printed, prints an object using the objects toString method and calls the prettyPrints setPrinted.
2. SetPrinted – Flips a boolean value 'printed' so that it wont be printed again.

Each object implementing this interface must add a boolean variable used in the setPrint and prettyPrint methods.

The boolean value:
The boolean values used in this implementation is either 1 for true or 0 for false.

Enums:
Directions and Operators are implemented as enums.

Interpretation of Robol:
The assignment is implemented such that each Robol Object interprets itself and calling other classes methods as needed. Like a Left statements interpreter calls the Robots moveLeft method with the number stored in the statement as parameter. Each interpreter() of the subclasses of a statement calls the parent/super' interpreter. The abstract statements interpreter() prints out the current statement with the prettyPrint method.

Other designs:

class Position: A Robol position, contains an x and y coordinate representing a legal coordinate inside a grid and a direction.

Class StatementList: Wrapper to hold multiple statements, the interpreter() interprets each statement int that list.

Storing of variable declarations: Variable declarations are stored in a hashmap, where the identifier is the key, and the variable declaration is the value. Since there only is one scope, I didn't think much about static or dynamic scoping.

Exceptions and error handling: I've added some exceptions which will throw RuntimeExpections, but there's room for a lot more error handling.

Description of the implementation of the Robol language

Program – The program takes a robot and grid as parameters to its constructor. It has three methods:

1. addStatement – adds a statement to the robot.
2. AddVarDecl - adds a variable declaration to the robot.
3. Interpreter - interprets the robot.

Grid – The grid takes a size as parameter to its constructor. Its only used to check if the robot is out of bounds and to print out the grid when the robot stops.

Size – Is just a container with two Numbers, used as a grids size.

X_/Y_Bound – Represented as two Numbers inside a Size object.

Robot – Contains two lists, one with variable declarations and one with statements and methods for each of the robots actions descriped in the assignment text.

Statement – An abstract class, parent for all statements, it contains the programs robot, so that each substatement can call the robots actions and the interpreter only prints out the current statement.

Start – Calls the robots setStartPosition.

Stop – Calls the robots stop method which will print out the robots position and print out the grid.

Move – Abstart super class of Left, Right, Forward. It constans the getValue() method which returns t the value of this statement as an integer. The interpreter only prints out this statement while each subclass' interpreter calls the super class' interpreter and also the Robots respective movement action (eg. MoveForward, moveLeft etc).

PenUp – Calls the robots penUp method which sets a boolean value (penDown) in the robot to false.

PenDown – Same as PenUp except the boolean value is set to true.

Assignment – Contains an expression and an identifier. The interpreter method sets the identifiers expression to be the expression stored inside the assignment.

IfThenElse – Contains a boolean expression and 1 or 2 StatementLists, ifStatements and elseStatements. The interpreter first evaluates boolean expression, depending on the value and if the object has an else statement list, the if/else statement list are interpreted.

While – Contains a boolean expression and a StatementList. The interpreter evaluates the boolean expression, while the expression evaluates to true, the StatementList is interpreted.

Direction – Implemented as an enum with values: LEFT, RIGHT, UP, DOWN.

VarDecl – Contains an expression and an identifier. It has a method setExpression which makes the identifier point at a new expression. Whenever this method is called, the identifiers boolean variable 'evaluated' is set to false. The new expression also needs to be interpreted incase the expression itself contains a reference to the identifier in this variable declaration. If not, the identifier in the expression will use the new expression of the identifier when evaluated, while the old value is the one to be used. The interpreter only prints out the variable declaration.

Expression – Super class of all expressions. Contains a boolean variable 'evaluated' which tells if the expression has been evaluated or not. An expression needs to be evaluated to be able to get the value of the expression, but if it already has been evaluated, the already evaluated value can be returned. When assigning a new expression to an identifier, this boolean variable needs to be reset incase the identifiers already has been evaluated. Another major thing about the expressions is that its interpreter method doesn't print the expression.

Identifier – Contains an unique string, the identifiers name, when the value of the identifier is requested, its value is extracted from the Robots GlobalVariableDeclarations, if the identifier hasn't been declared, an error is thrown.

Number – Contains an integer.

PlusExpression – Contains two expressions and an operator, when requesting its value for the first time, the expression is interpreted. First the two expressions are interpreted, then depending on the operator the value variable is set to be the result of the evaluated expression and that operater.

(exp) – Not implemented, didn't see it before I started writing this text. It is probably used for precedens, but that doesn't seem very useful in this implementation, if you want precedents, you just have to construct the PlusExpression in the correct way.

BooleanExpression – Contains two expressions and an relation operator. When evaluated for the first time, the two expressions are evaluated before the expressions value is set by using the operator on the two expressions. If the expressions is true, the value is set to 1, else the value is set to 0.