EECS 3311 – Project Report

Fall 2019

TEAM  
Sarwat Shaheen (214677322)

Nafis Ahmed Awsaf (215306095)

EECS Logins of team members:

sarwat12

awsaf11

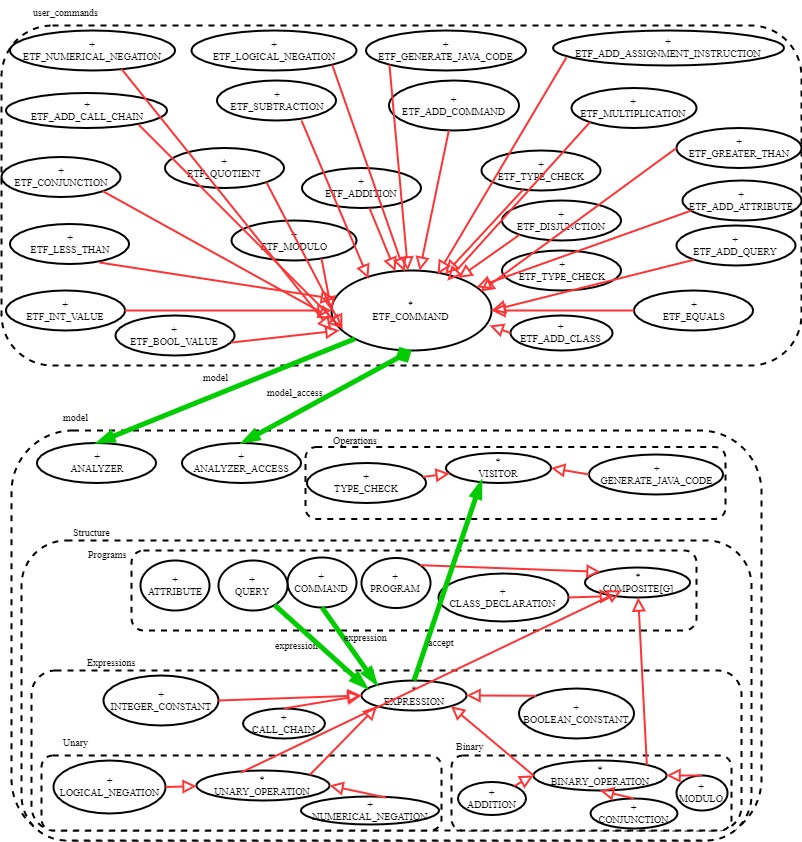
EECS login of submitting account:

sarwat12

BON DIGRAMS OF OUR DESIGN

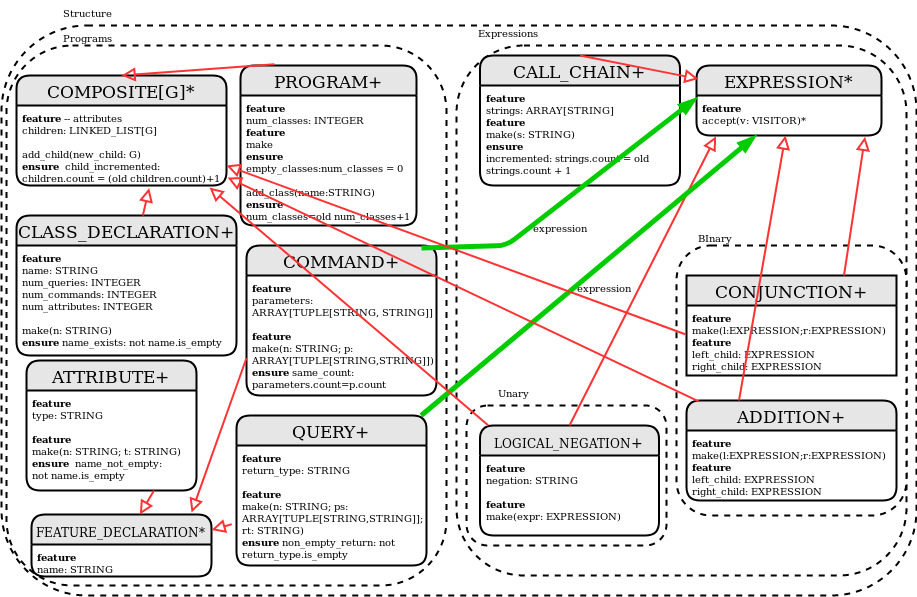
BON DIAGRAM 1

The following diagram details the relationship between all relevant classes within the project implementation. All classes in this diagram are shown in concise/compact view.



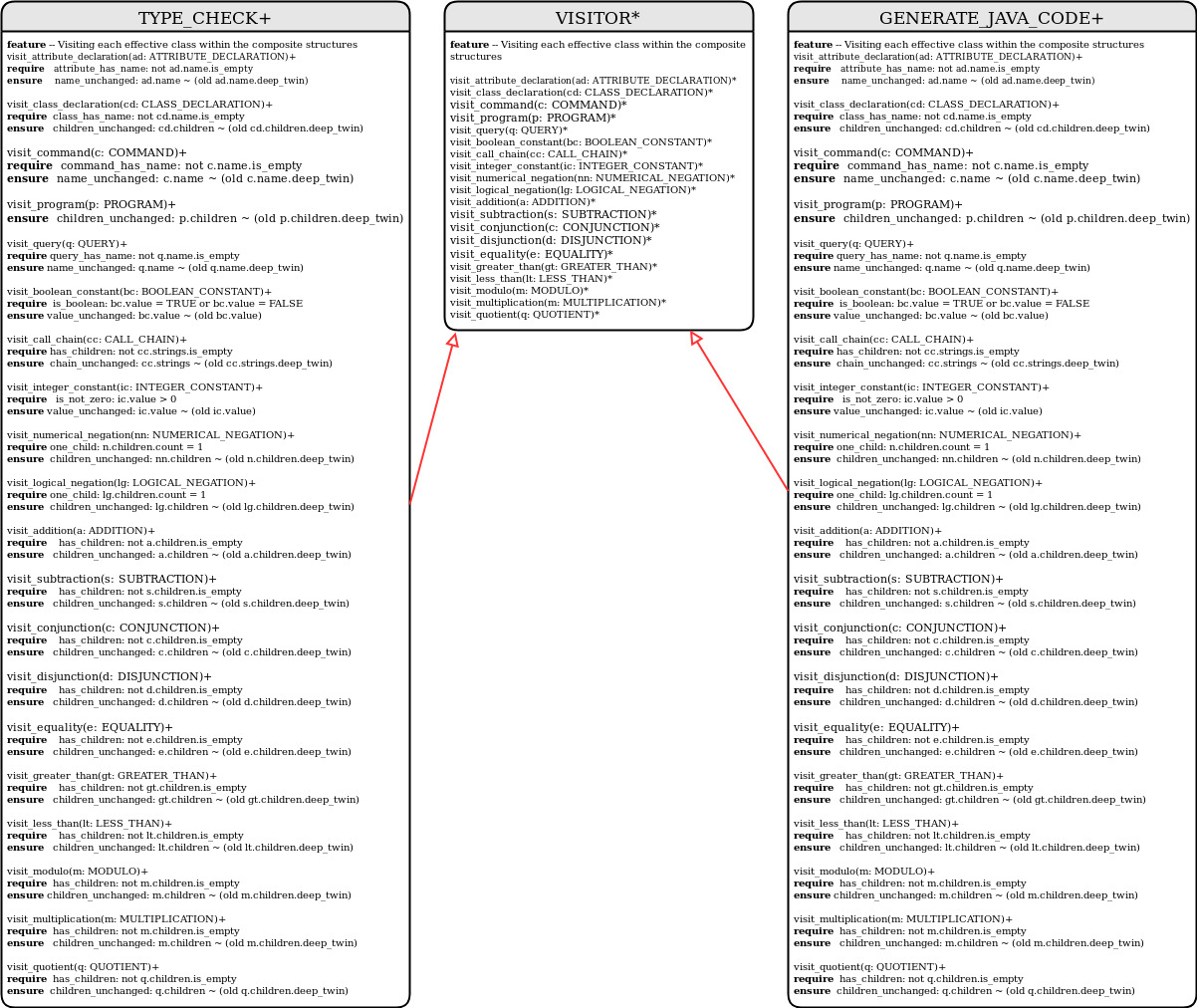
BON DIAGRAM 2

The following diagram details the architecture between all relevant classes within the programming language/expressions structure. Critical classes are shown in expanded/detailed view.



BON DIAGRAM 3

The following diagram details the architecture between all relevant classes within the programming language/expressions operations. Critical classes are shown in expanded/detailed view.



Details of how our design obeys the following design principles:

– Information Hiding

Ans: The principle of information hiding states the principle of separation of the design decisions within a class or project and making specific implementation features hidden from the client’s perspective. In our project, the critical class where we implemented information hiding principle is in the ANALYZER class, where the constructor make is hidden from the rest of the classes in the project, except ANALYZER\_ACCESS. This enables specific implementation details to be hidden from the client, ETF users, and other classes within the structure/operations cluster.

**Hidden and may be changed**: Underlying implementation in all the classes of operations/structure along with ANALYZER.

**Not hidden and stable**: The abstract user interface commands available to the clients as specified by the child classes of the ETF\_COMMAND

– Single Choice Principle

Ans: The Single Choice Principle states that whenever a program needs to support a set of specified features, they should all be included in one class, so that any future changes to such features take place in one module rather than going through changes separately.

The primary instance of the single choice principle is demonstrated in the COMPOSITE[G] where the children: LINKED\_LIST[G] is declared along the feature add\_child. Any child classes which inherit from COMPOSITE[G] further avoids re-declaring children: LINKED\_LIST. If the underlying implementation of child changes from using a LINKED\_LIST to an ARRAY, that change can be made only in the COMPOSITE[G] class, without the need for changing any subsequent child classes.

Another instance of obeying the single choice principle in our project is in the FEATURE\_DECLARATION class where the name: STRING feature is declared which is then inherited by the ATTRIBUTE, COMMAND, and QUERY classes respectively. Any further changes in this feature would only need to be dealt with in the FEATURE\_DECLARATION class, and duplicate changes may be avoided.

– Open-Close Principle

Ans: The open-closed principle states that some parts of our project should be open for modification and other parts should be closed from further changes.

An instance of obeying the open-closed principle in our project in within the composite and visitor patterns used in the overall implementation.

The program structure and expressions cluster is closed for modification, which means adding any further structural classes would violate the single choice principle and would require making multiple changes in the visitor pattern implementation.

The program operations which are supported by the project through the use of the visitor pattern is open for modification. New operations can be added and will be supported by the program structure/expressions without having to violate single-choice principle.

– Uniform Access Principle

Ans: The Uniform Access Principle states that the notation of features used throughout a class should not change regardless of whether it is an attribute or a routine.

An instance of obeying this principle can be found in the ANALYZER class where the **out** feature, responsible for pretty printing the current state of the program, can be uniformly called without having worrying about the feature being an attribute or a routine. The same principle can also be found in the program\_status and class\_status features in the classes PROGRAM and CLASS\_DECLARATION respectively.

– Any other design principle that we discussed in lectures

\* Principles of polymorphism and dynamic-binding have been used in the CLASS\_DECLARATION class, where the children array is declared initially of static type FEATURE\_DECLARATION. But when accessed dynamically, the objects might be either an ATTRIBUTE, COMMAND, or a QUERY, making the children a polymorphic LINKED\_LIST.

\* Principles of cohesion has also been obeyed in our project through the use of the visitor pattern. The program operations supported in the project are grouped together according to similar functionality and kept separate from the program structure. The operation implementations are kept in separate classes in order to only perform their respective tasks.

\*Principles of Generics have also been used in the project, primarily in the COMPOSITE[G] class, where clients of the class can specify a composite structure of any time just by inheriting it. In our project, class PROGRAM is a composite of CLASS, and BINARY\_OPERATION is a composite of expressions. Both classes are able to specify their composite types just through the use of inheritance and generics, allowing for re-usable code.