Project PolyClass

# Purpose

The purpose of this program is to be able to convert a class in one language into a class in any other programming language. This program will create a simple framework for the design of the class in the two languages. It will not incorporate the logic and/or inner body of the class or methods nor values of any member variables. A class that can be translated into any language shall be called a PolyClass©. The PolyClass© is based on <http://hyperpolyglot.org/> and the fact that the structure for most classes is incredibly similar.

# Elements of the Program

[FileTokenizer](#_The_LanguageTokenizer) An abstract class that reads a file and tokenizes from a file into the program.

[LanguageAnalyzer](#_The_LanguageParser) An abstract class that analyzes tokens from the file using [FileTokenizer](#_The_LanguageTokenizer).

[PolyClass](#_The_PolyClass©)© Contains all the information needed to create a class in any language. Note that some languages support features that others do not; these members are simply ignored (i.e. duck typing).

[LanguageAssembler](#_The_LanguageInterpreter) An interface. A [PolyClass](#_The_PolyClass)© object may be passed into the constructor for any LanguageAssembler which will then use the [FileWriter](#_The_LanguageWriter_1) to create the file of the given type.

[FileWriter](#_The_LanguageWriter_1) Writes the class to file given from any [LanguageAssembler](#_The_LanguageAssembler).

Supported Languages Each language should have its own classes that inherit from either a [FileTokenizer](#_The_LanguageTokenizer) and [LanguageAnalyzer](#_The_LanguageAnalyzer) for reading a language or just a [LanguageAssembler](#_The_LanguageInterpreters) for writing the language. This project should be able to support the creation of a tokenizer, analyzer and assemblers for any Object Oriented language. *Examples* below:

* Programming Languages
  + C++ (.h and .cpp)
  + Java (.java)
* Modeling / Markup Languages
  + HTML (.html or .htm)
  + UML (.txt)
  + XML (.xml)
* Scripting Languages
  + Javascript (.html or .htm)
  + Python (.py)
* Other Languages
  + perl
  + SQL (.txt) - modeling

# The FileTokenizer

A FileTokenizer will be independent of how a language is structured. The tokenizer will simply return individual token Strings from the file via the nextToken() and previousToken().

|  |
| --- |
| FileTokenizer |
| - operators : String  - symbols : String  - indentLevel : long  - line : long  - fileScanner : Scanner  - fileInput : FileInputStream  - tokens : LinkedList<String>  $ stack : Stack<String> |
| FileTokenizer(filepath : String)  - nextLine() : String  + previousToken() : String  + nextToken() : String  $ stackIsBalanced() : boolean  $ stackReset() : void  $ stackUpdate(input : String) : boolean |

# The LanguageAnalyzer

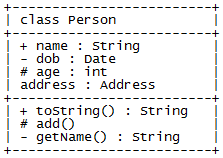
A LanguageAnalyzer should be able to analyze from the tokens the proper visibility, modifiers, identifiers, etc. from surrounding tokens to appropriate the data for every class and its members and pass it into a [PolyClass](#_The_PolyClass©) (5) object. Analyzers should ignore the body of a function and work with only the structure of a class. For example, in a Java program, an opening '{' after an operation declaration should indicate the analyzer to ignore data (i.e. keep reading but do nothing with the data) until the respective closing '}' is found.

|  |
| --- |
| *LanguageAnalyzer* |
| - tokenizer : [FileTokenizer](#_The_LanguageTokenizer)  *- keywords : LinkedList<String>*  *- comment : String*  *- mlCommentOpen : String*  *- mlCommentClose : String*  *- mlComment : boolean*  - polyClass : [PolyClass](#_The_PolyClass©) |
| + LanguageAnalyzer(tokenizer : FileTokenizer)  *+ analyzeFile() : void*  *+ analyzeClass() : void*  *+ analyzeAttribute(): void*  *+ analyzeOperation(): void*  *+ analyzeArguments(): void*  *+ analyzeVisibility(): void*  *+ analyzeModifier(): void*  *+ analyzeIdentifier(): void*  *+ analyzeDataType(): void*  *+ getPolyClass() : PolyClass* |

To determine where an operation or a class ends (for example Java-like languages that have beginning and terminating braces {}) use a Stack to check for balanced parentheses, brackets, and braces.

## Plain Text UML Analyzer (Example LanguageAnalyzer)

The UMLAnalyzer (as modeled after ***Figure 1***) need only call analyzeFile() method. The analyzeFile() method should then incorporate all the below methods recursively.

analyzeClass()

* check if the line ends with "<T>", or "<E>", etc. this indicates the class uses generics
* check if next line is "<<Interface>>"
* check if next line is "<<Enumeration>>" or "<<enum>>"
* check if next line is "<<Abstract>>" (normally abstract items should be italicized, but this is not possible in plain text
* check if line begins with "$" or next line is "<<Static>>" (normally abstract items should be underlined, but this is not possible in plain text

Figure

* As good coding style indicates, one should have attributes declared before operations, but a well coded program should be able to identify when there is an operation or attribute. So for now call analyzeAttributes() and then call analyzeOperations().

analyzeAttributes() - if line begins with "+-" return

analyzeAttribute() - should call each of the following once:

* analyzeVisibility()
* analyzeIdentifier()
* analyzeDataType()

analyzeOperations() - if line begins with "+-", or "--" return

analyzeOperation() - If the operation identifier is the same as the Class name, this is a constructor (not void return type)

analyzeVisibility() - could be '+' (public), '-' (private), '#' (protected), '$' (static), or none

analyzeModifier()

analyzeIdentifier()

analyzeDataType() - matches ": String", ": int", ": double", etc.

cleanline()

* Remove all the '-', '+', and '|' characters
* Find the index of ':', '(', and ')' characters
* Trim whitespace from the line

# The PolyClass©

The PolyClass© is a generic class (unrelated to Java Generics) that can be used to define all the properties of a class as defined in any Object-Oriented programming language. The following parts make up the PolyClass©. Class models are listed in order of which they are required by reference.

## The PolyClassPart

The abstract PolyClassPart contains variables and methods that are common to all Classes, Attributes, and Operations.

|  |
| --- |
| *PolyClassPart* |
| # identifier : String  # mfr\_static : boolean  # mfr\_constant : boolean  # mfr\_abstract : boolean  # visibility : int |
| Get and set for each attribute above as appropriate. |

Visibility attribute should follow this scheme: 0-none, 1-private, 2-protected, 3-public.

## The PolyDataType

The PolyDataType is an enumeration that will define the different primitive data types common to most Object-Oriented languages. It will also help to differentiate between primitive data types and custom classes.

|  |
| --- |
| PolyDataType  <<Enumeration>> |
| VOID  BOOLEAN  CHARACTER  BYTE  SHORT  INTEGER  LONG  FLOAT  DOUBLE  STRING  CLASS |

The PolyDataType.CLASS value will be a reference to a specific external class that may be a custom or built-in library class (like MyCustomClass or Java’s Scanner). There then will be a member of the Attribute, Argument, Return, or Operation called className which will be a String value naming the specific class as seen in the [PolyAttribute](#_The_PolyAttribute) (5.3), [PolyArgument](#_The_PolyOperationPart) (5.4), [PolyReturn](#_The_PolyReturn) (5.5), and [PolyOperation](#_The_PolyOperation) (5.7) classes.

## The PolyAttribute

Also known in some languages as a member variable or field, the PolyAttribute will inherit from the [PolyClassPart](#_The_PolyClassPart) (5.1) and have the following structure.

|  |
| --- |
| PolyAttribute |
| - className : String  - datatype : PolyDataType |
| + PolyAttribute(identifier : String, className : String)  + PolyAttribute(identifier : String, dataType : PolyDataType)  Get and set for each attribute above as appropriate. |

It is recommended that a PolyAttribute be instantiated anonymously.

## The PolyArgument

The PolyArgument will be arguments or parameters for a [PolyOperation](#_The_PolyOperation) (5.6) and will have the following structure. It will inherit from the [PolyReturn](#_The_PolyReturn) (5.5).

|  |
| --- |
| PolyArgument |
| - identifier : String |
| + PolyArgument(identifier : String, className : String)  + PolyArgument(identifier : String, dataType : PolyDataType)  Get and set for each attribute above as appropriate. |

The setClassName() will automatically fill the dataType field to PolyDataType.CLASS (5.2). It is recommended that a PolyArgument be instantiated anonymously within a [PolyOperation](#_The_PolyOperation) (5.6) object.

## The PolyReturn

The PolyReturn will be return objects for a [PolyOperation](#_The_PolyOperation) (5.6) and will have the following structure. It will *not* inherit from the [PolyClassPart](#_The_PolyClassPart_1) (5.1).

|  |
| --- |
| PolyArgument |
| - dataType : PolyDataType  - className : String |
| + PolyReturn(className : String)  + PolyReturn(dataType : PolyDataType)  Get and set for each attribute above as appropriate. |

It is recommended that a PolyReturn be instantiated anonymously within a [PolyOperation](#_The_PolyOperation) (5.6) object.

## The PolyOperation

Also known in some languages as a member methods or functions. The PolyOperation will inherit from the [PolyAttribute](#_The_PolyClass) (5.3) and have the following structure.

|  |
| --- |
| PolyOperation |
| arguments : LinkedList<[PolyArgument](#_The_PolyOperationPart)>  returns : LinkedList<[PolyReturn](#_The_PolyReturn)>  isConstructor : boolean  isDestructor : boolean |
| getNumArgs() : int  addArg(argument : [PolyArgument](#_The_PolyOperationPart)) : void  getArgList() : LinkedList<[PolyArgument](#_The_PolyOperationPart)>  getNumReturns() : int  addReturn(argument : [PolyReturn](#_The_PolyReturn)) : void  getReturnList() : LinkedList<[PolyReturn](#_The_PolyReturn)>  Get, has, and set for each attribute above as appropriate. |

The addArg() and addReturn() methods will add arguments or return objects in the order they are called and will not be rearranged, reordered, or sorted.

## The PolyClass

The PolyClass will inherit from the [PolyClassPart](#_The_PolyClassPart) (5.1) and have the following structure.

|  |
| --- |
| PolyClass |
| - polyClassVersion : int  - isInterface : boolean  - interfaces : LinkedList<String>  - parentClass : LinkedList<String>  - attributes : LinkedList<[PolyAttribute](#_The_PolyAttribute)>  - operations : LinkedList<[PolyOperation](#_The_PolyOperation)>  - innerClasses : LinkedList<[PolyClass](#_The_PolyClass)>  + hasInnerClass(className : String) : boolean  + hasOperation(opName : String) : boolean  + hasAttribute(attrName : String) : boolean |
| Get, has, and set for each attribute above as appropriate. |

The interfaces variable indicates the interface classes (Java Interfaces) which this instance implements.

# The LanguageAssembler

A LanguageAssembler is a class that should use the proper coding style and formatting for the language it will represent. It will have the following structure.

|  |
| --- |
| LanguageAssembler  <<Interface>> |
| - indentLevel : long  - scopeLevel : long  - writer : [FileWriter](#_The_LanguageWriter)  - classToAssemble : [PolyClass](#_The_PolyClass©) |
| + LanguageAssembler(classToTranslate : [PolyClass](#_The_PolyClass))  + assembleClass() : void  - assembleAttributes() : void  - assembleAttribute() : void  - assembleOperations() : void  - assembleOperation() : void  - assembleVisibility() : void  - assembleModifier() : void  - assembleIdentifier() : void  - assembleDataType() : void |

Language assemblers should pass lines into the [FileWriter](#_The_LanguageWriter_1) (7) that are formatted appropriately according to the specific language’s proper coding style.

## Sample Code Translations

Some examples of translating code below:

|  |  |  |  |
| --- | --- | --- | --- |
| Concept | Java[[1]](#footnote-1) | C++[[2]](#footnote-2) | Python[[3]](#footnote-3) |
| Primitive Data Types | byte  boolean  char  int  long  double  float | byte  bool  char  int  long  double  float | (duck typing) |
| String | String | std::string | (duck typing) |
| Attribute and Operation Visibility | None  private  protected  public | None (means public)  private  protected  public | (all visible)  Internal parts usually begin with a \_ in the identifier name. |
| Attribute and Operation Modifiers | final  static  abstract | const  static  virtual | - |
| Inner Classes | Yes | Yes | Yes |
| Aggregate Objects (1 object within an object) | Yes | Yes | Yes |
| Composite Objects (0+ objects within an object) | Yes | Yes | Yes |
| Class Declaration | *visibility* **class** ClassName**{** | *visibility* **class** ClassName**{** | **class** ClassName**:** |
| Class Inheritance | *visibility* **class** ChildClass **extends** ParentClass { | **class** ChildClass **:** *visibility* ParentClass { | **class** ChildClass**(**ParentClass**):** |
| Comment | // | // | # |
| Multiline Comment | /\*\*/ | /\*\*/ | ''' three single quotes ''' |
| Generics | **class** MyGenericsClass**<**T**>** | template | class \_ClassName: |
| Return Types | Void + Primitive Data Types | | - |
| No. of Return Values | One | One | Infinite |

# The FileWriter

The FileWriter is a simple file write class that uses java.io.FileOutputStream and java.io.PrintWriter to write to file lines that are passed into it by a [LanguageAssembler](#_The_LanguageAssembler_1). It will have the following structure.

|  |
| --- |
| FileWriter |
| - filepath : String  - file : FileOutputStream  - printer : PrintWriter |
| + open(filepath : String) : boolean  + writeLine(line : String) : void  + close() : boolean |

# References

Attributes, Java http://docs.oracle.com/javase/tutorial/reflect/class/classMembers.html

Class Modifiers, Java http://docs.oracle.com/javase/tutorial/reflect/class/classModifiers.html

Class, UML http://pages.cs.wisc.edu/~hasti/cs302/examples/UMLdiagram.html

https://www.tutorialspoint.com/uml/uml\_basic\_notations.htm

Class, XML RELAX NG http://www.relaxng.org/tutorial-20011203.html

Class, XML RELAX NG Compact http://www.relaxng.org/compact-tutorial-20030326.html

Enumeration, UML http://stackoverflow.com/questions/412944/uml-class-diagram-enum

Generics, Java http://docs.oracle.com/javase/tutorial/java/generics/types.html

Language Interpretation http://hyperpolyglot.org/

Nested Classes, C++ http://en.cppreference.com/w/cpp/language/nested\_types

Visibility, Java http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

1. <https://docs.oracle.com/javase/7/docs/api/java/lang/Class.html> [↑](#footnote-ref-1)
2. <http://www.cplusplus.com/doc/tutorial/classes/> [↑](#footnote-ref-2)
3. <https://docs.python.org/2/tutorial/classes.html> [↑](#footnote-ref-3)