Serious Game
Developer Specifications

March 14, 2013

Abstract

Contents

1 Call Stack Application Implementation

1.1 Back-End

The main classes that form the back-end of the Call Stack application are Variable, Poignee, Environnement and ParseurXML.

Each instance of the Variable class will contain an instance of a genuine Java type with its name in the environment. An instance of the Variable class will also contain all of the Variables to which it is linked. A variable 1 is linked to another variable 2 if the value of one of variable 2 's fields is variable 1. In that case, both variable 2 and the name of all of the fields that have variable 1 as their value will be contained within variable 1.

Each instance of Poignee will be a pointer that will contain its name, its type and the variable to which it is currently pointing.

An instance of the Environment class will contain all of the Variable instances as well as all of the Poignee instances in two ArrayList. It will also containt the name of all the types that are considered to be primitive and all of the types that will be displayed by the front-end. The methods of the Environment class are used :

- to call Constructors on Variables, create Variables containing all the instances created by the Constructor and add them to the Environment
- to invoke Methods on Variables of the Environment and take into account all the side effects and the instances that are potentially created
- to change the values of the fields of the Variables by assigning them to other Variables or Fields of Variables
- to change the Variables to which the Poignee instances are pointing

An instance of ParseurXML will parse a XML configuration file specifying the names of the classes to be drawn and the primitive classes. By using the Class.forName() method, the Java types will be created from their names in the XML file in order to then be used in the Environment.

1.2 Front-end

The main classes that form the front-end are GUI, ChoixArguments, RetourMethode and DrawEnvironnementBis

The GUI class will contain :

- the interface that is mainly made up of JComboBoxes and JButtons
- an instance of Environment in its envCourant field
- an instance of DrawEnvironnementBis in its paintbis field that will display the diagram representing all of the instances of the Environment contained in the envCourant field

The ChoixArguments class is a JFrame that will pop up whenever a Constructor or a Method that requires at least an argument has been called. It will allow the user to select the value of each argument among the available Variables of compatible type. Moreover, whenever an argument has a primitive type, the user will be able to type in directly its value.

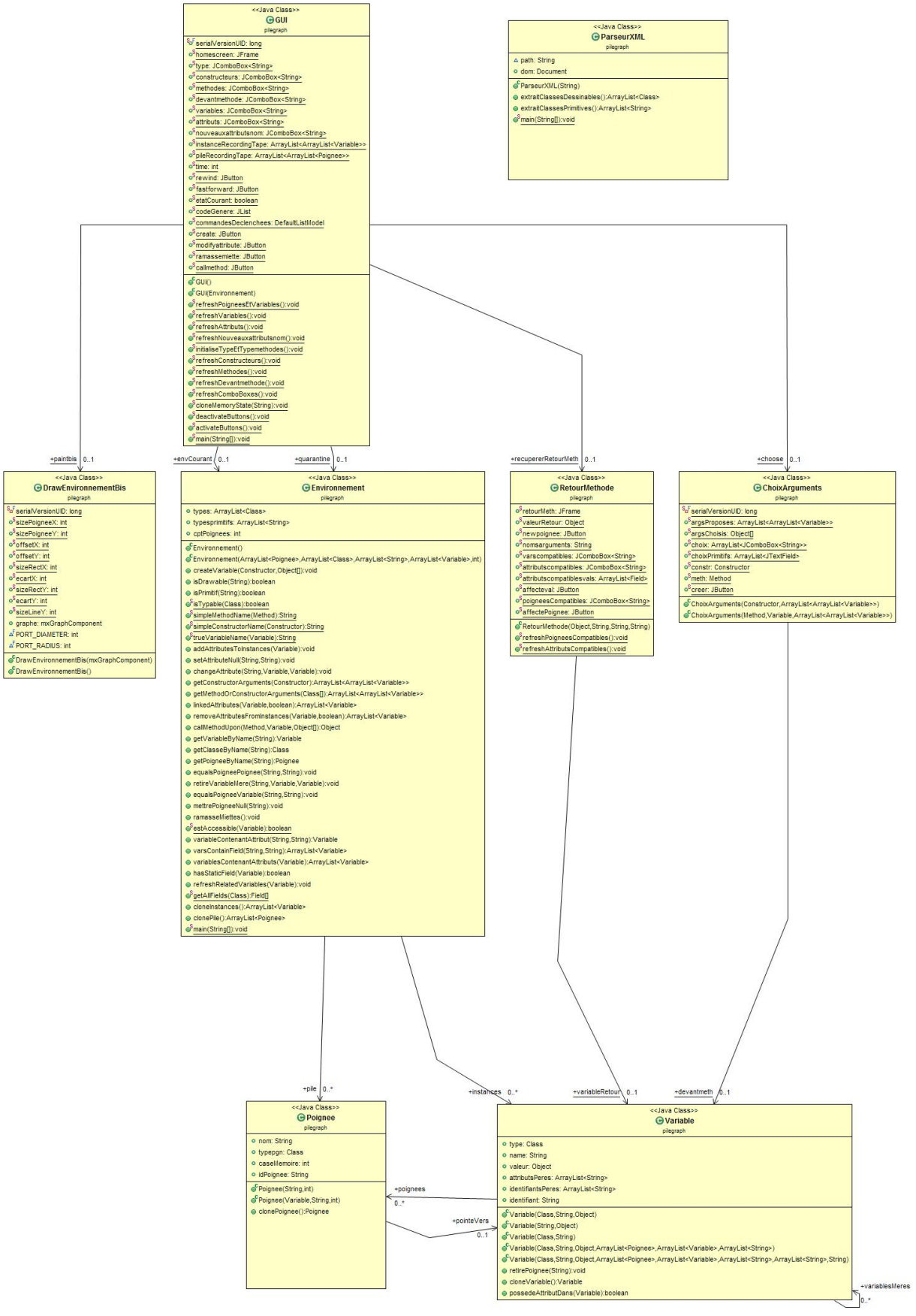
The RetourMethode class contains a JFrame that will pop up whenever a Method that returns a non void value has been invoked. It will allow the user to choose how to assign this return. The return can be:

- assigned to a new pointer
- assigned to a pre-existing pointer
- assigned to the field of a Variable

The DrawEnvironnementBis class uses the JGraphX library and especially the Ports of the vertices created in order to display :

- the vertices representing the pointers
- the vertices representing the variables
- the links between the pointers and their values
- the links between the fields of each Variable and their values which are other Variables

1.3 UML Diagram



1.4 Technical Specification

Requ ID	Functionality	Description
1	Variable class	Storage of the value, name and type of a variable
2	Variable class	When a variable 2 is the value of the x field of a variable 1, the variable 1 and the name of the x field will be respectively stored in the fields variablesMeres and attributsPeres of the variable 2
3	createVariable, getMethodOrConstructorArguments, ChoixArguments class, addAttributesToInstances	Creation of a new instance of any given type. The instance and its fields are then added to the environment
4	changeAttribute, setAttributeNull	Assignment of a field to the null value or to a new value contained within another variable of the environment
5	callMethodUpon, getMethodOrConstructorArguments, ChoixArguments class, linkedAttributes, addAttributesToInstances	Invoke methods on a variable, take into account the side effects of the method, the new fields that could potentially be created are added to the environment
6	cloneMemoryState, trueVariableName, the fields codeGenere & commandesDeclenchees of the GUI class	After each significant operation of the user, the corresponding Java code is saved in a JList in order to display it. The names of the pointers in the call stack will be displayed instead of the names of the variables in the memory.
7	cloneMemoryState, the fields instanceRecordingTape, pileRecordingTape, time & quarantine of the GUI class	After each significant operation of the user, the environment is cloned and stored in a list. A time variable is incremented after each operation of the user.
8	isDrawable, isPrimitif	An instance will only be displayed if its type appears among the drawable types specified in the XML file. Moreover, the fields of an instance will be added to the environment only if the instance's type isn't primitive.
9	hasStaticField, refreshRelatedVariables, addAttributesToInstances, linkedAttributes	If a created variable or a variable on which a method has been invoked has a static field, all of the links originating from the instances of the same type as the variable, its supertypes and its subtypes are removed and then re-established.
10	ParseurXML Class	XML configuration file containing the names of the drawable classes and the primitive classes will be parsed. The resulting data will be stocked in the environment.

1.5 Versions Delivered

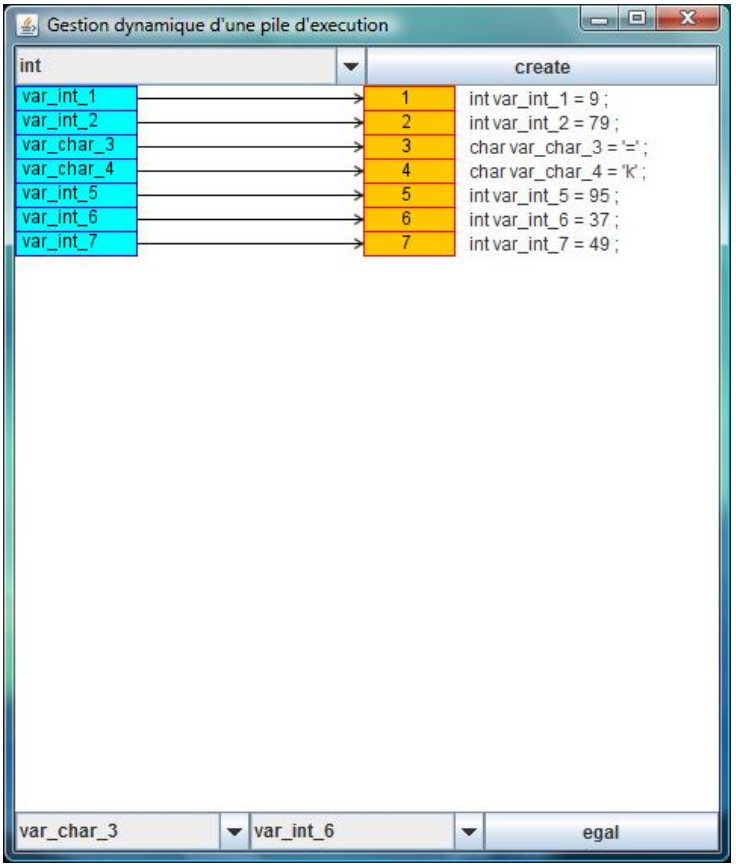


Figure 1: Version 0.5

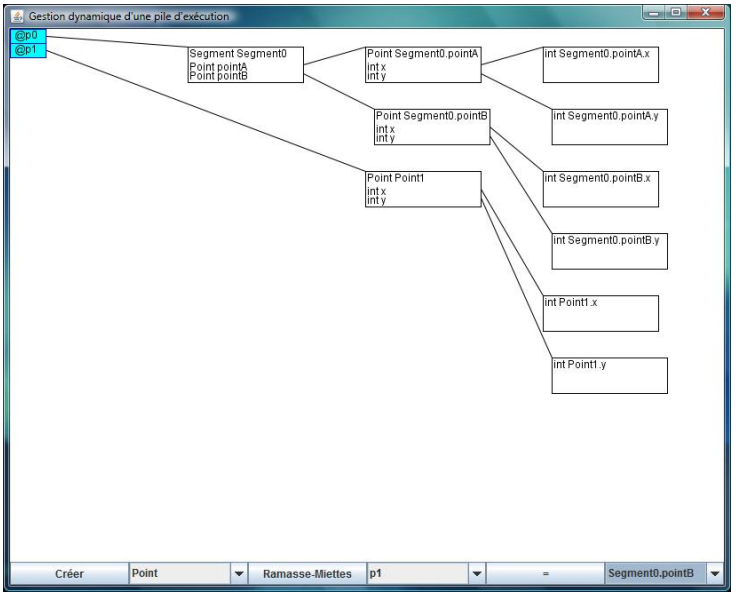


Figure 2: Version 1.0

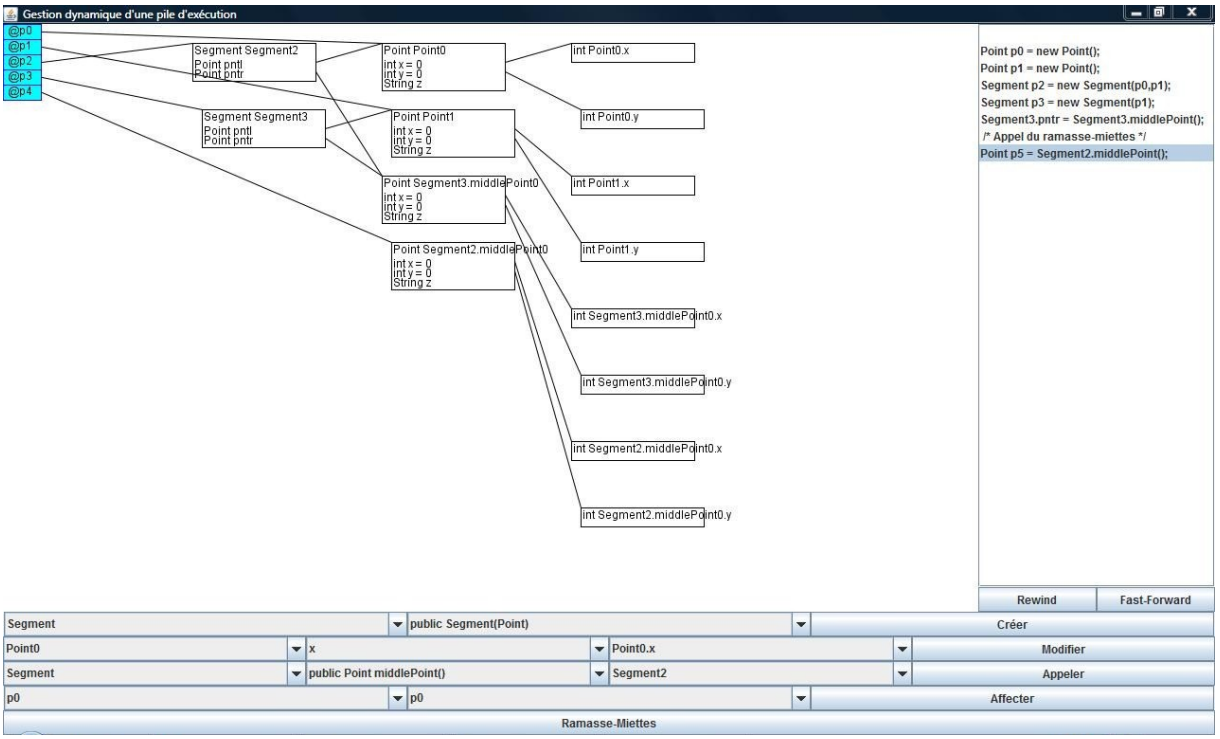


Figure 3: Version 2.0

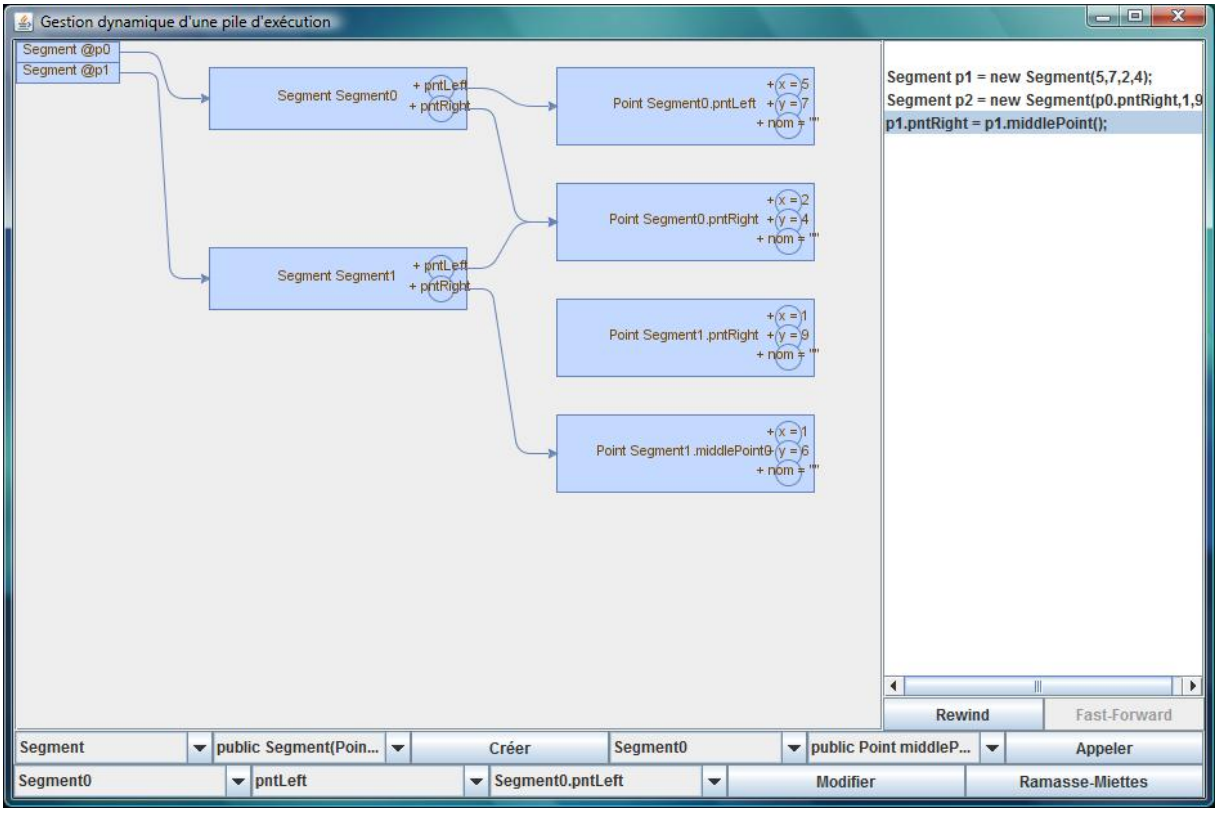


Figure 4: Version 3.0

2 UML Diagram Application Implementation

2.1 Back-End

The main classes that form the back-end of the UML Diagram application are Liaison, Etiquette, CustomKeyBoardHandler, mxCellEditor, mxGraph and ParseurXML.

Each instance of Liaison contains two instances of Etiquette : one for the source and the other for the target.

An instance of Liaison also contains four Strings : two for the roles and the two others for the multiplicities that are associated to the source and the target Etiquettes of Liaison.

Each instance of Etiquette contains two Strings one for the name and the other for the Id. An instance of Etiquette contains also an integer for the rank with which the node can be correctly positioned on the UML diagram and finally a list of fields associated to the Class or the Etiquette.

The Class ParseurXML contains methods that extract data from XML files to construct the correct Etiquettes and Liaisons thanks to the methods getEtiquettes(), getLiaisons(), and the text that will be displayed to the user for helping him to construct the good UML Diagram thanks to the method getText().

The Classes of JGraphx have been modified among them the class mxGraph in the package com.mxgraph.view where the method cellsRemoved(Object[] cells) have been completed to remove correctly links from the "liaison_cree" when the user press the keyboard button "SUPPR". The other class modified is named mxCellEditor in the package com.mxgraph.swing.view where the method stopEditing(boolean) have been completed to manage the change of cells label once the user has already linked the cell with a first name and then want to change the label of the cell for any reason. This modification was done in order to make the application robust and able to work in abnormal conditions.

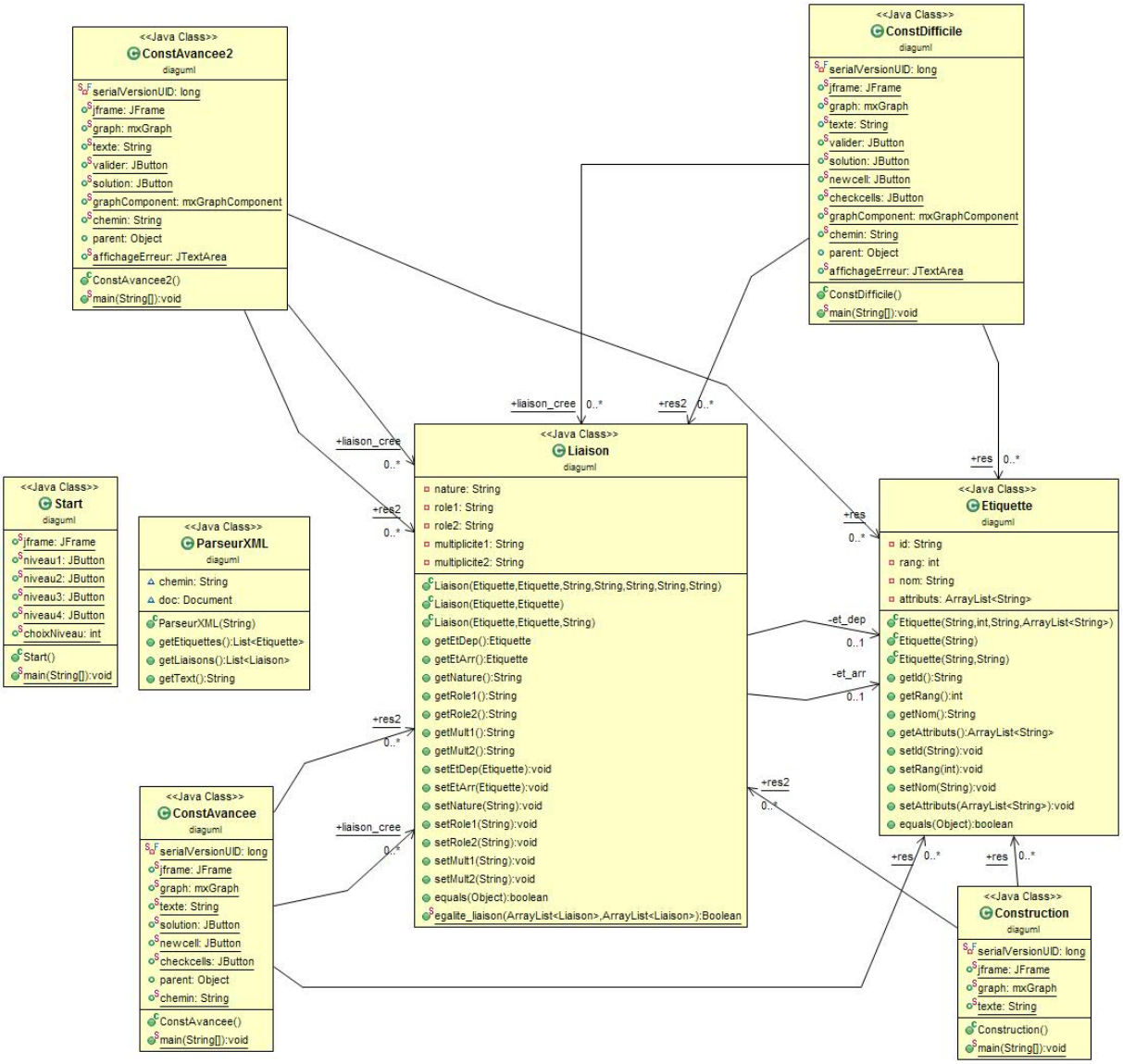
2.2 Front-End

The main classes that form the front-end are Start, Construction, ConstAvancee, ConstAvancee2, ConstDifficile and Solution.

The Class Start play the role of the menu where we can choose the level of difficulty of diagram construction. It contains four buttons, behind each one there are the classes Construction, ConstAvancee, ConstAvancee2 and ConstDifficile.

The classes Construction, ConstAvancee, ConstAvancee2 and ConstDifficile contain the scenarios of construction of UML diagrams from the easiest to the hardest. Those four classes heavily use the JGraphx Library to construct the diagram. In this library the vertices represent the classes and the edges represent the links between those classes.

2.3 UML Diagram



2.4 Technical Specification

Requ ID	Functionality	Description
1	`ParseurXML` class, getEtiquettes(), getLiaisons(), getText()	Extraction of the significant elements of the XML documents concerning classes or interfaces as well as the relations between them.
2	The launch menu, `Start` class	Menu allowing the user to choose the difficulty level
2.1	Construction class	Graphical Interface containing the usage scenario of the first difficulty level : placing the links between the correctly positioned nodes.
2.2	ConstAvancee class	Graphical Interface containing the usage scenario of the second difficulty level : giving each correctly positioned node its correct label and placing the links between the nodes.
2.3	ConstAvancee2 class	Graphical Interface containing the usage scenario of the third difficulty level : placing the links between the labelled nodes that are randomly scattered across the window.
2.4	ConstDifficile class	Graphical Interface containing the usage scenario of the fourth difficulty level : creating the correct number of nodes, labelling them by reading a descriptive text and linking them together.
3	`Text` Tag Text in XML file, ParseurXML.getText(), JTextArea textarea, classes 2.*	Presence of a JTextArea component containing the text to be displayed on the upper part of each window of any difficulty level.
4	JOptionPane.showMessageDialog at the level of classes, `ConstAvancee` & `ConstAvancee2`	Dialog boxes giving the user indications concerning the labels of the nodes or information about the correct construction of the UML diagram.
5	fields `role1` & `role2` of class `Liaison` , getRole1(),getRole2(), setRole1(string s), setRole2(string s), classes 2.* Methode : graphComponent.getConnectionHandler().addListener()	The roles in the composition, aggregation and association relations are taken into account.
5.1	Class `Liaison` field `nature`, getNature(), setNature(String nat), classes 2.* Methode : graphComponent.getConnectionHandler().addListener()	The type of the link between two nodes is taken into account.
5.2	Class `Liaison` field `nature`, getNature(), setNature(String nat), classes 2.* Methode : graphComponent.getConnectionHandler().addListener()	The type of the link between two nodes is taken into account.
5.3	Class `Liaison` field `nature`, getNature(), setNature(String nat), classes 2.* Methode : graphComponent.getConnectionHandler().addListener()	The type of the link between two nodes is taken into account.

Requ ID	Functionality	Description
6	Class `Liaison` field `nature`, getNature(), setNature(String nat), classes 2.* Methode : graphComponent.getConnectionHandler().addListener()	The type of the link between two nodes is taken into account.
7	JButton valider, JButton checkcells, Classe Solution	Interaction with the user in order to validate the actions accomplished before clicking on the button, check the labels of the nodes and display the solution on a new window afficher la bonne solution sur une nouvelle fenêtre.
8	Class `Liaison` fields `multiplicite1`, `multiplicite2`, getMult1(), getMult2(), setMult1(string s), setMult2(string s), classes 2.* Method : graphComponent.getConnectionHandler().addListener()	Multiplicities can be associated to links between classes which will enable the user to better understand relations in a UML Diagram
9	Class `Etiquette` field `ArrayList attributs`, getAttributs()	The classes can contain fields that will characterise them more.

2.5 Difficulty Levels

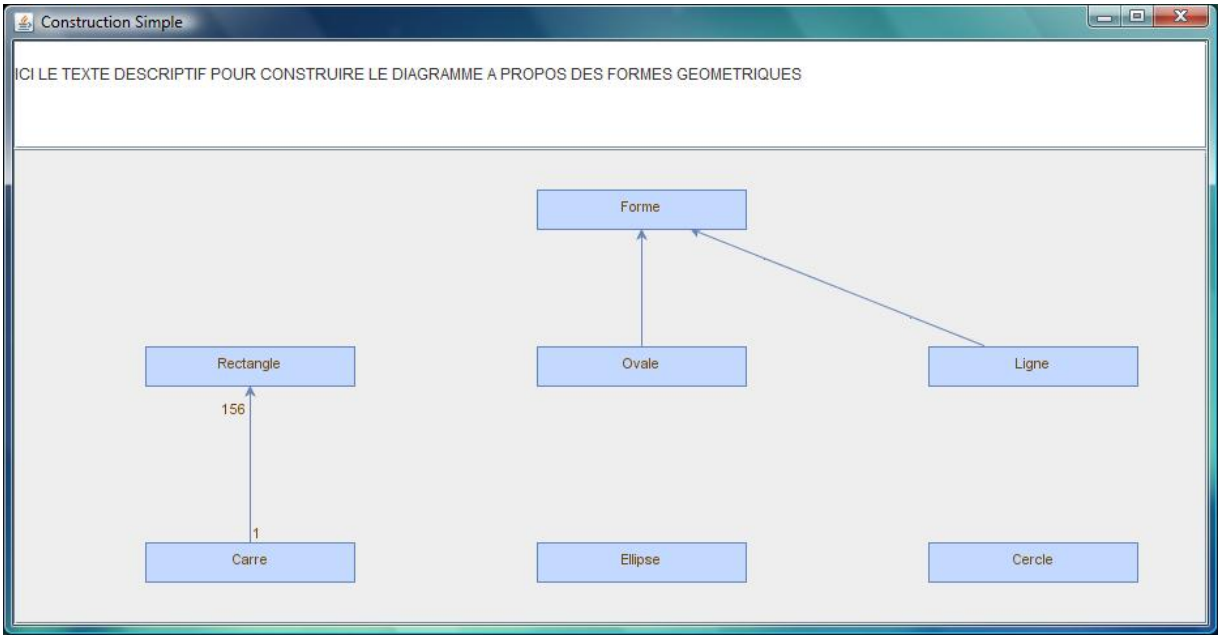


Figure 5: Easiest Difficulty Level

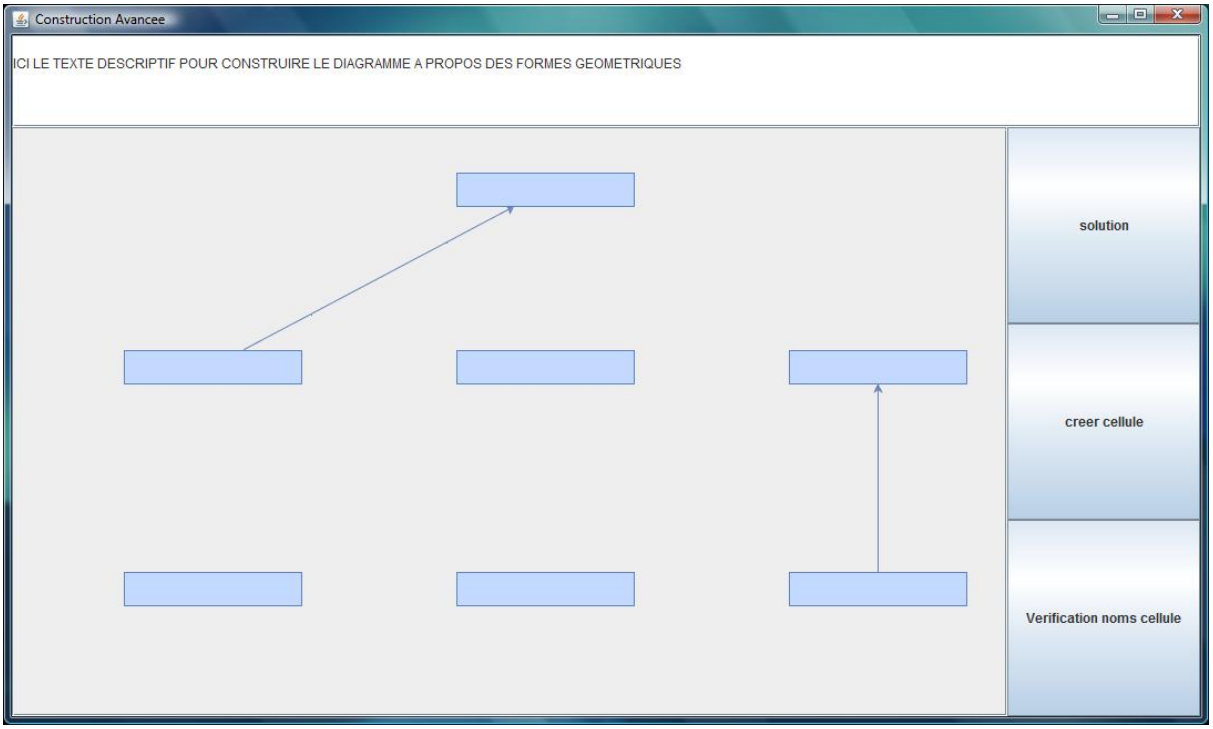


Figure 6: Intermediate Difficulty Level

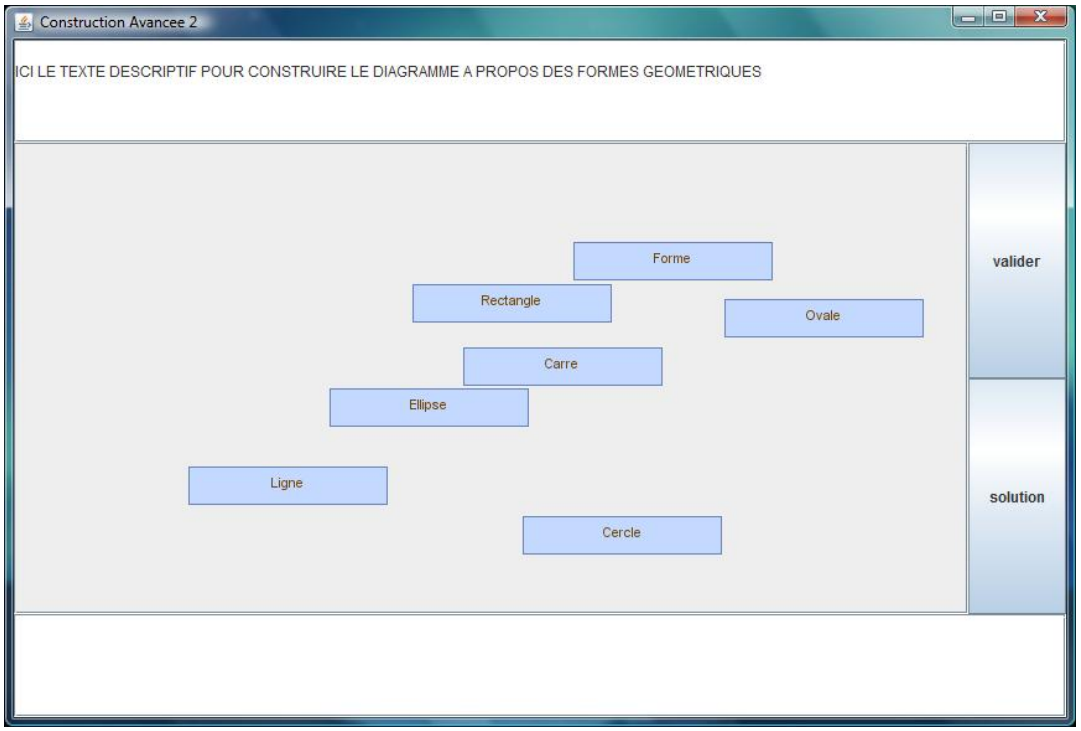


Figure 7: Higher Difficulty Level

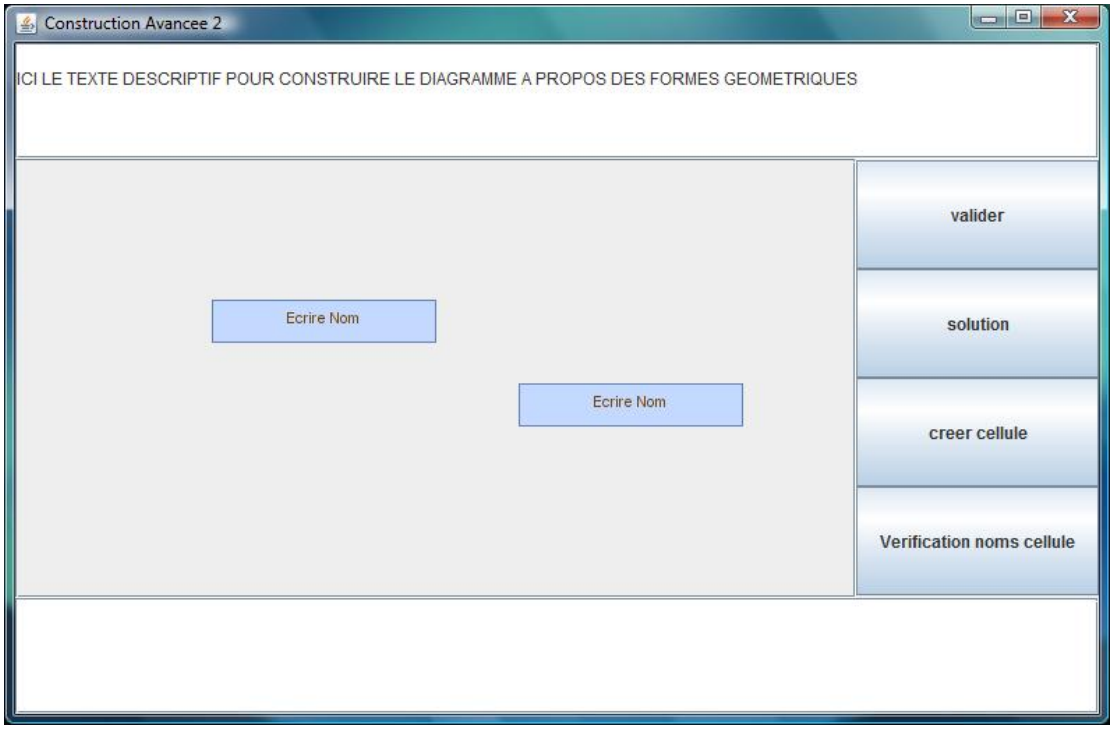


Figure 8: Hardest Difficulty Level

3 Instantiation and Field Modification Application Implementation

The design of this application had to remain as simple as possible. In order to achieve that, the number of classes that were implemented had to remain as low as possible.

3.1 Back-end

The back-end is composed of the single class Variable which contains an instance of a Java Class and its name in the environment.

3.2 Front-end

The front-end comprises the classes GUI, DrawDiagram, DrawInstance and Modification.

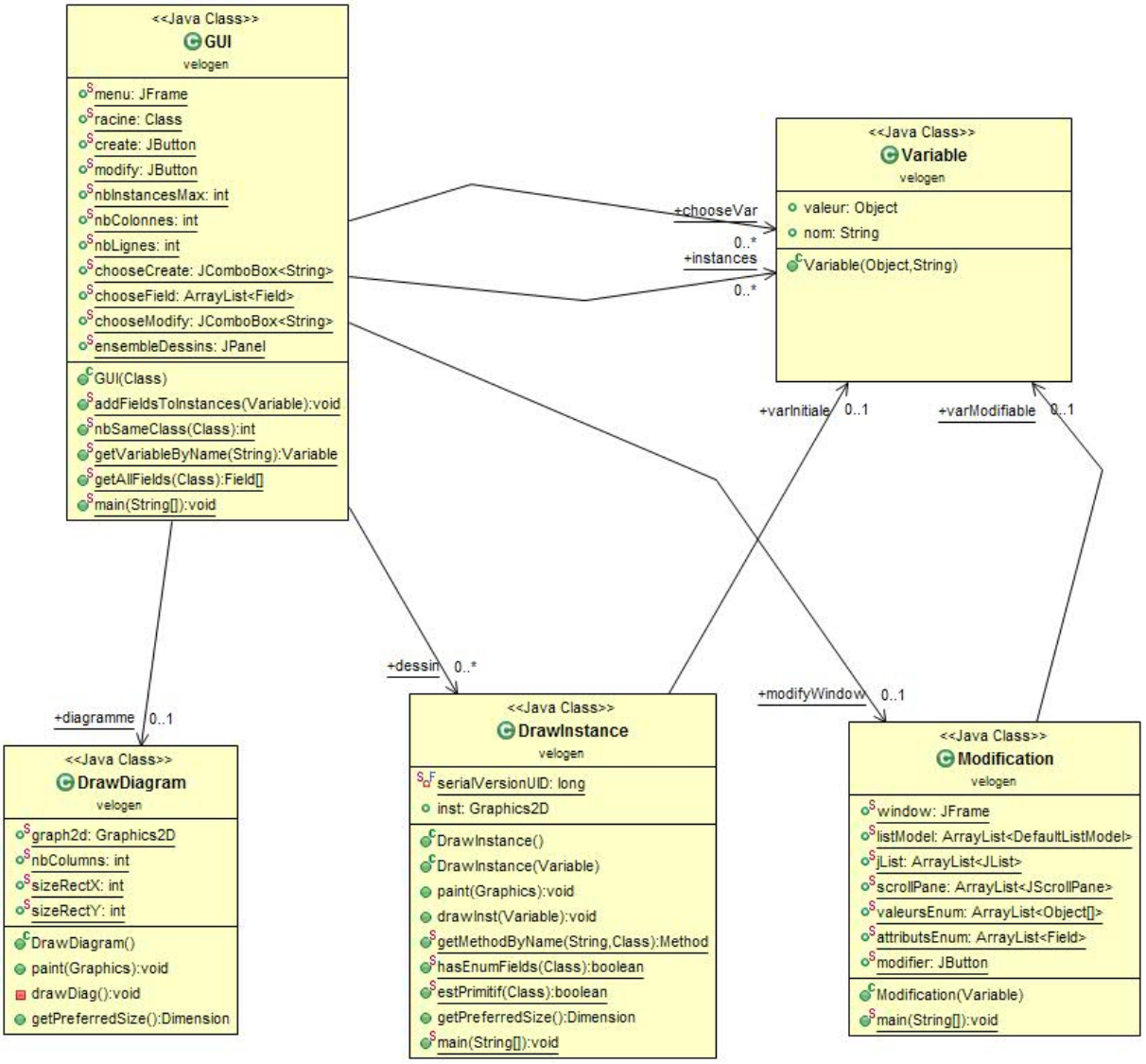
`GUI` contains in its menu field the main menu that will appear when the application is launched. The interface in composed of two JComboBoxes, two JButtons, an instance of DrawDiagram in its diagram field, a list of DrawInstance in its dessin field.

`Modification` contains a JFrame that will pop up whenever the user wants to modify the value of the Enum fields of a variable. The window will list for each Enum field the possible values and the current value of the Enum field will be highlighted. Once the user has chosen the new value of each Enum field, by clicking on a "Modify" button, the new values will be taken into account and the fields of the Java instances stored in Variables will be modified.

`DrawDiagram` will display the names of the created instances in the environment and the values of each of their fields. If a field has an Enum type, its value will be directly displayed. If the field's type is neither an Enum type nor a primitive type, the name of the variable that contains the value of the field will be displayed.

`DrawInstance` will visually display a picture representing a created instance. The picture will change according to the value of the fields with Enum types.

3.3 UML Diagram



3.4 Technical Specification

Requ ID	Functionality	Description
1	Introspection API java.lang.reflect, Enum types	The fields that the user will be able to modify will have an Enum type. Recovering the possible values of Enum types, creating instances and modifying their fields will be done through the Java Introspection API java.lang.reflect
2	`DrawInstance` class	Affichage d'une instance principale, puis appel récursif afin d'afficher chacun de ses attributs de type non enum. Chaque fonction d'affichage dépendra des valeurs des attributs de type enum de l'instance.
3	JButton entitled `create` of the `GUI` class, `addFieldsToInstances` method	Instantiation of a class through java.lang.Constructor by using the first constructor that doesn't require any arguments. Its fields are then added to the environment recursively.
4	`Modification` class	Choice of the new value of each field among all the possible values of an Enum type. The value of the fields are then modified through the class java.lang.reflect.Field
5	`DrawDiagram` class	All the created variables are visually displayed on instances of the Graphics2D class alongside the name and field values of each variable which are determined through the java.lang.reflect.Field class

3.5 Final Version 2.0

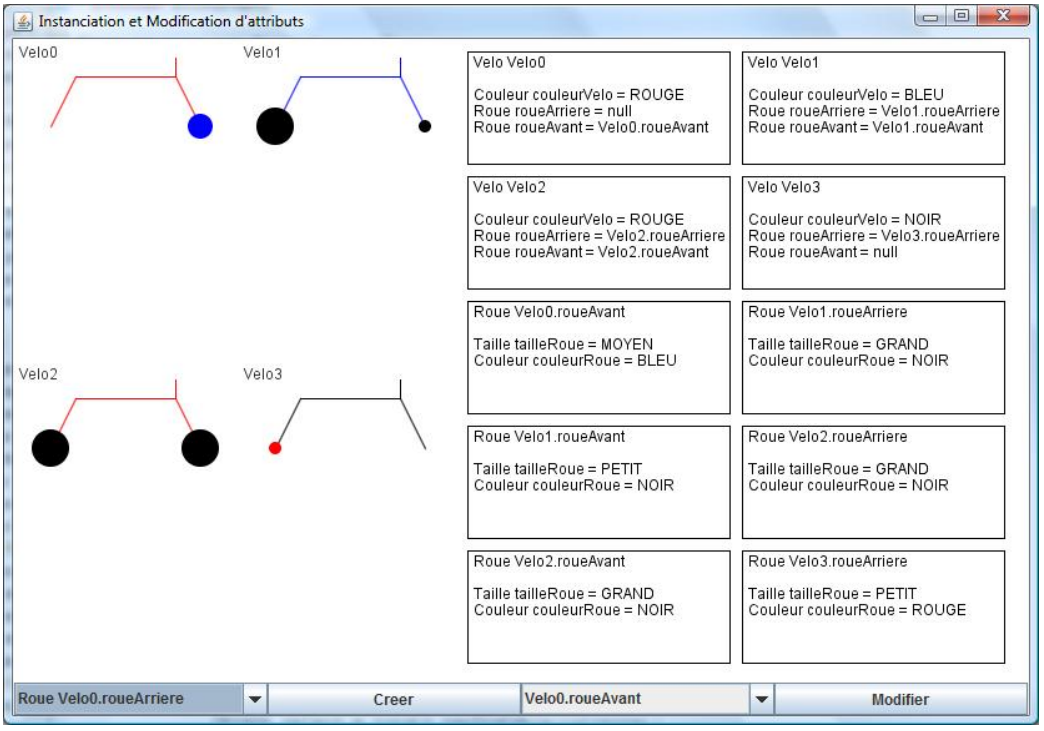


Figure 9: Main Menu

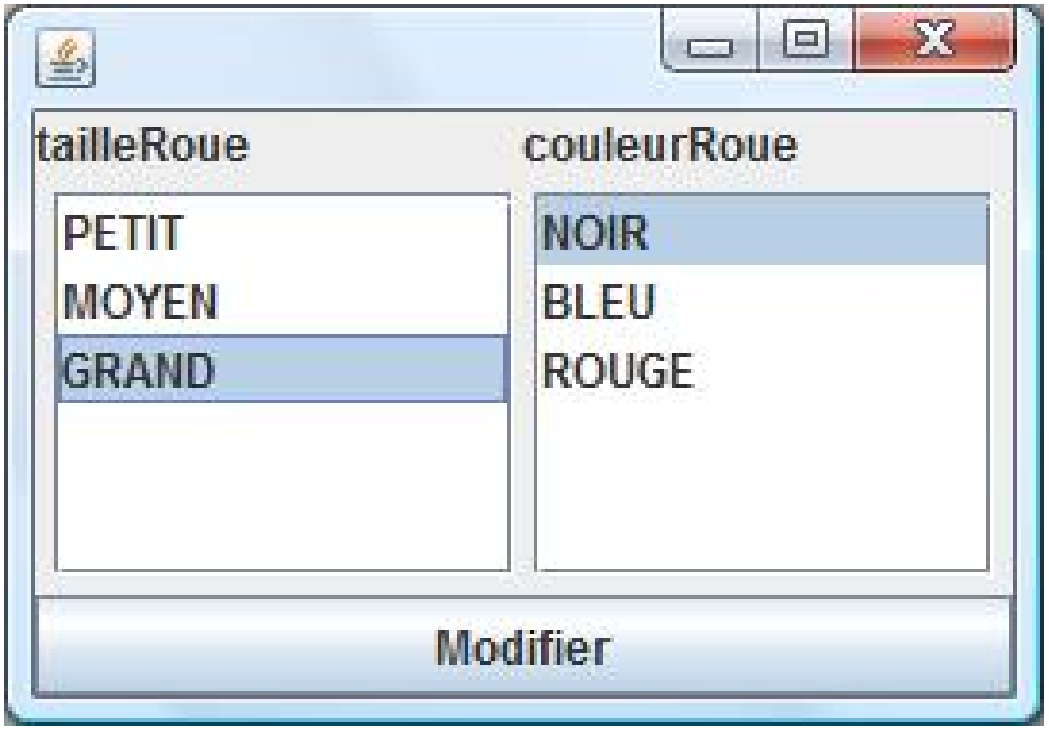


Figure 10: Modification of Fields

4 Call Stack Application Javadoc

Since our customers requested the javadoc and the user manuals in French, they have been written in French in compliance with our customers' demands.

4.1 ChoixArguments.java

Fenetre qui apparaitra lorsque l'utilisateur devra choisir la valeur de chaque argument d'une methode ou constructeur appele parmi les variables de type compatible de l'environnement

4.1.1 Fields

`argsProposes` : argsProposes.size() = argsChoisis.length = nombre d'arguments du constructeur ou de la methode appelee pour tout i entre 0 et argsProposes.size()-1, argsProposes.get(i) contiendra toutes les variables de GUI.envCourant compatibles avec le i-eme argument du constructeur ou methode appelee

`argsChoisis` : Pour tout i entre 0 et argsChoisis.length - 1, argsChoisis[i] contiendra la valeur de chaque variable que l'utilisateur aura choisie comme argument a passer au constructeur ou a la methode appelee

`choix` : Pour tout i entre 0 et argsProposes.size()-1, si le i-eme argument du constructeur ou methode appelee ne peut pas etre fourni directement par l'utilisateur en le tapant avec le clavier, les noms de chaque variable de argsProposes.get(i) seront stockees dans une JComboBox qui sera ensuite ajoutee a l'attribut choix

`choixPrimitifs` : Pour tout i entre 0 et argsProposes.size()-1, si le i-eme argument du constructeur ou methode appelee peut etre fourni directement par l'utilisateur en le tapant avec le clavier, une JTextField sera creee et ajoutee a l'attribut choix afin que l'utilisateur y ecrit la valeur qu'il souhaite fournir

`constr` : si un constructeur a ete appele, constr contient le constructeur appele

`meth` : si une methode a ete appelee, meth contient la methode appelee

`devantmeth` : si un contructeur a ete appele, devantmeth contient la variable sur laquelle la methode a ete appelee

`creer` : des que tous les arguments seront choisis, la methode ou le constructeur sera appelee en cliquant sur le JButton creer

4.2 DrawEnvironnementBis.java

graphe representant l'environnement constitue de poignees et de variables, chaque poignee pouvant pointer vers une variable et chaque attribut de variable pouvant pointer vers une autre variable qui est sa valeur

4.2.1 Fields

`sizePoigneeX` : longueur des noeuds representant une poignee

`sizePoigneeY` : largeur des noeuds representant une poignee

`offsetX` : offset horizontal de tous les noeuds representant des variables

`public static int offsetY` : offset vertical de tous les noeuds representant des variables

`public static int sizeRectX` : longueur des noeuds representant une variable

`public static int ecartX` : ecart horizontal entre deux noeuds representant des variables

`public static int sizeRectY` : largeur d'un noeud representant une variable

`public static int ecartY` : ecart vertical entre deux noeuds representant des variables

`public static int sizeLineY` : Epaisseur accordee a chaque ligne sur laquelle un attribut d'une variable sera ecrit. La largeur d'un noeud representant une variable sera (nombre d'attributs + 2)*sizeLineY

`public mxGraphComponent graphe` : representation graphique de l'environnement, avec chaque poignee et variable representee par un noeud, et les liens entre les noeuds indiquant les valeurs des poignees ainsi que les valeurs des attributs de chaque variable. Le mxGraphComponent sera ajoute dans la JFrame homescreen de GUI afin que l'utilisateur puisse voir graphiquement l'etat courant de l'environnement

`final int PORT_DIAMETER` : diametre du noeud representant l'attribut d'une variable. Chaque attribut sera un noeud ayant pour noeud parent le noeud representant la variable possedant cet attribut. Selon la terminologie de JGraphX, chaque attribut sera donc un port du noeud representant la variable possedant cet attribut.

`final int PORT_RADIUS` : rayon du noeud representant l'attribut d'une variable. Chaque attribut sera un noeud ayant pour noeud parent le noeud representant la variable possedant cet attribut. Selon la terminologie de JGraphX, chaque attribut sera donc un port du noeud representant la variable possedant cet attribut.

4.3 Environnement.java

4.3.1 Fields

`public ArrayList<Poignee> pile` : ensemble des poignees de l'environnement

`public ArrayList<java.lang.Class> types` : classes dont l'utilisateur pourra creer des instances

`public ArrayList<String> typesprimitifs` : types consideres comme etant primitifs

`public ArrayList<Variable> instances` : ensemble des variables creees

`public int cptPoignees` : nombre de poignees dans l'environnement courant. Cet entier servira a generer les noms des variables creees.

4.3.2 Methods

`public void createVariable(Constructor constr, Object[] cstrargs)` : creation d'une variable a partir d'un constructeur et des valeurs de ses arguments, puis stockage de la variable creee dans l'environnement

`public boolean isDrawable(String classname)` : renvoie true si et seulement si classname figure parmi les classes contenues dans l'attribut types de l'environnement. La classe sera alors affichee dans le graphe de DrawEnvironnementBis

`public boolean isPrimitif(java.lang.String classname)` : renvoie true si et seulement si classname figure parmi les noms de classes contenus dans l'attribut typesprimitifs de l'environnement. La classe est alors consideree comme etant primitive.

`public static boolean isTypable(Class type)` : renvoie true si et seulement si l'utilisateur peut taper manuellement la valeur d'une variable du type fourni en argument. Les types pouvant etre fournis manuellement sont Integer, int, Double, double et String

`public static String simpleMethodName(Method meth)` : renvoie le nom simple de la methode constitue de : visibilite de la methode, nom simple du type de retour, nom de la methode, puis nom simple du type de chaque argument

`public static String simpleConstructorName(Constructor constr)` : renvoie le nom simple du constructeur constitue de : visibilite du constructeur, nom du type cree, puis nom simple du type de chaque argument

`public static String trueVariableName(Variable var)` : renvoie le nom d'une variable de l'environnement debutant par le nom d'une poignee a laquelle elle est liee soit directement, soit indirectement en passant par des attributs d'autres variables, comme par exemple poignee1.attributA.attributB

`public void addAttributesToInstances(Variable var)` : ajoute dans l'environnement tous les attributs de la variable var qui sont non primitifs et non nulls et qui ne sont pas deja presents dans l'environnement, puis appel recursif de addAttributesToInstances sur tout attribut non primitif et non null, peu importe qu'il soit deja present ou non dans l'environnement

`public void setAttributeNull(String nomVar, String nomAtt)` : mise a null de l'attribut nomAtt de la variable nomVar

`public void changeAttribute(String attributeName, Variable hasAttribute, Variable newAttribute)` : modifie la valeur de l'attribut attributeName de la variable hasAttribute en lui affectant pour valeur la variable newAttribute

`public ArrayList<ArrayList<Variable>> getConstructorArguments(Constructor cstr)` : renvoie pour chaque argument d'un constructeur toutes les variables de type compatible dans l'environnement stockes dans une liste. Comme pour chaque argument une ArrayList sera determinee, le retour sera une ArrayList¿

`public ArrayList<ArrayList<Variable>> getMethodOrConstructorArguments(Class[] typesArguments)` : renvoie pour chaque argument d'un constructeur ou d'une methode toutes les variables de type compatible dans l'environnement stockes dans une liste. Comme pour chaque argument une ArrayList sera determinee, le retour sera une ArrayList¿

`public ArrayList<Variable> linkedAttributes(Variable culprit, boolean start)` : supprime les liens entre une variable et tous ses attributs et realise ensuite la meme chose recursivement a chacun de ces attributs qui est de type non primitif. Renvoie la liste des attributs supprimes

`public ArrayList<Variable> removeAttributesFromInstances(Variable culprit, boolean start)` : renvoie la liste des attributs d'une variable culprit. On ajoute ensuite recursivement a cette liste les attributs de chaque attribut.

`public java.lang.Object callMethodUpon(java.lang.reflect.Method meth, Variable upon, Object[] methargs)` : callMethodUpon appelle la methode meth avec les arguments methargs sur la variable upon. Comme la methode peut avoir des effets de bord sur la variable et ses attributs, on supprime les liens entre la variable et ses attributs en appelant la methode linkedAttributes . Les attributs de la variable seront ensuite rajoutes a l'environnement apres l'appel de la methode meth sur upon en appelant la methode addAttributesToInstances. si la methode possede un attribut declare en static, afin de prendre en compte la modification potentielle de cet attribut qui se fera sur toutes les instances possedant ce meme attribut en static, il faudra supprimer puis rajouter les attributs de chaque variable possedant cet attribut static. La methode refreshRelatedVariables realise ceci.

`public Variable getVariableByName(String namevar)` : recupere de l'environnement la variable ayant pour nom namevar

`public Class getClasseByName(String classname)` : recupere de l'environnement la classe ayant pour nom classname

`public Poignee getPoigneeByName(String namepgn)` : recupere de l'environnement la poignee ayant pour nom namepgn

`public void equalsPoigneePoignee(java.lang.String namepgnleft, String namepgnright)` : affecte la poignee ayant pour nom namepgnleft la valeur pointee par la poignee ayant pour nom namepgnright

textttpublic void retireVariableMere(String nomattribut, Variable varMere, Variable varFille) : retire des attributs variablesMeres et attributsPeres de la variable varFille respectivement la variable varMere et la chaine nomattribut

`public void equalsPoigneeVariable(String namepgn, String namevar)` : affecte la poignee ayant pour nom namevar a la variable nommee namevar de l'environnement

`public void mettrePoigneeNull(String namepgn)` : affecte la poignee nommee namepgn a la valeur null

textttpublic void ramasseMiettes() : le ramasse-miette supprime de l'environnement les variables qui sont completement inaccessibles, ie aucune poignee ne permet d'y acceder, que ce soit directement ou indirectement par le biais des attributs des variables vers lesquelles les poignees pointent.

`public static boolean estAccessible(Variable var)` : renvoie true si et seulement si une variable

est accessible, que ce soit directement a travers une poignee quand cette poignee pointe vers la variable ou a travers les attributs des variables vers lesquelles les poignees pointent

`public Variable variableContenantAttribut(String nomVarMere, String nomAttPere)` : renvoie la variable de l'environnement contenant l'attribut nomAttPere de la variable NomVarMere

`public ArrayList<Variable> varsContainField(String nomVarMere, String nomAttPere)` : renvoie toutes les variables ayant pour variable mere la variable nommee nomVarMere et pour attribut pere correspondant l'attribut nomAttPere. Fonction de test permettant de verifier qu'a tout moment, tout attribut non null d'une variable quelconque pointe sur une unique variable.

`public ArrayList<Variable> variablesContenantAttributs(Variable var)` : renvoie toutes les variables filles de la variable var, ie toutes les variables vers lesquelles les attributs non nulls de var pointent

`public boolean hasStaticField(Variable var)` : verifie recursivement si une variable ainsi que toutes les variables ayant pour valeurs les attributs de cette variable possedent au moins un attribut declare en statique

`public void refreshRelatedVariables(Variable var)` : soit TypeA le type de la variable var. refreshRelatedVariables supprime les liens entre les variables ayant pour type TypeA, tout supertype de TypeA ou sous-type de TypeA. Les attributs de toutes ces variables sont ensuite rajoutes a l'environnement en appelant la fonction addAttributesToInstances sur chacune de ces variables.

`public static Field[] getAllFields(Class type)` : renvoie toutes les variables declarees dans la classe type et toutes ses superclasses

`public ArrayList<Variable> cloneInstances()` : renvoie une copie de toutes les variables de l'environnement. Les copies des instances ne seront pas affectees par les modifications effectuees sur les variables de l'environnement originel

`public ArrayList<Poignee> clonePile()` : renvoie une copie de toutes les poignees de l'environnement. Les copies des poignees ne seront pas affectees par les modifications effectuees sur les poignees de l'environnement originel

4.4 GUI.java

4.4.1 Fields

`public static Environnement envCourant` : l'environnement affiche sera contenu dans l'attribut envCourant. Lorsque l'utilisateur retournera en arriere dans le temps, l'environnement courant sera stocke dans l'attribut quarantine et l'attribut envCourant sera affecte a un environnement precedent

`public static DrawEnvironnementBis paintbis` : paintbis est une instance de DrawEnvironnementBis qui affichera - les poignees et variables contenues dans l'environnement - les liens entre chaque poignee et la valeur vers laquelle elle pointe - les liens entre chaque variable et la valeur de chacun de ses attributs qui pourront etre d'autres variables

`public static JFrame homescreen` : JFrame qui sera affichee lorsque l'application sera executee

`public static JComboBox<String> type` : JComboBox contenant le nom simple de tous les types dont l'utilisateur pourra creer des instances

`public static JComboBox<String> constructeurs` : JComboBox dans laquelle tous les noms simples des constructeurs du type selectionne dans la JComboBox type seront affiches

`public static JComboBox<String> methodes` : JComboBox dans laquelle les noms simples des methodes de la classe selectionnee dans la JComboBox type seront affichees

`public static JComboBox<String> devantmethode` : JComboBox stockant les noms de toutes les variables de type compatible avec celui selectionne dans la JComboBox type

`public static RetourMethode recupererRetourMeth` : attribut qui contiendra la fenetre qui apparaitra lorsque l'utilisateur devra affecter un retour non void d'une methode

`public static JComboBox<String> variables` : JComboBox qui contiendra les noms des variables et poignees de l'environnement courant

`public static JComboBox<String> attributs` : Lorsqu'une variable est selectionnee dans la JComboBox variables, les noms de tous ses attributs seront affiches dans la JComboBox attributs. Lorsqu'une poignee est selectionnee, cette JComboBox sera videe de son contenu.

`public static JComboBox<String> nouveauxattributsnom` : Lorsqu'une variable est selectionnee dans la JComboBox variables et qu'un attribut est selectionne dans la JComboBox attributs, les noms des variables de type compatible avec l'attribut selectionne dans la JComboBox attributs seront affichees dans la JComboBox nouveauxattributsnom. Lorsqu'une poignee est selectionnee dans la JComboBox variables, la JComboBox attributs sera videe de son contenu et les noms des variables et poignees de type compatible avec la poignee selectionnee dans la JComboBox variables seront affiches dans la JComboBox nouveauxattributsnom

`public static ArrayList<ArrayList<Variable>> instanceRecordingTape` : l'attribut instanceRecordingTape contiendra les copies de l'ensemble des variables de l'environnement faites suite a toute operation significative de l'utilisateur telle que l'appel d'un constructeur, d'une methode ou la modification d'un attribut

`public static ArrayList<ArrayList<Poignee>> pileRecordingTape` : l'attribut pileRecordingTape contiendra les copies de l'ensemble des poignees de l'environnement faites suite a toute operation significative de l'utilisateur telle que l'appel d'un constructeur, d'une methode ou la modification d'un attribut

`public static int time` : l'attribut time est une variable temps qui s'incrementera suite a toute operation significative de l'utilisateur telle que l'appel d'un constructeur, d'une methode ou la modification d'un attribut

`public swing.JButton rewind` : le JButton rewind permettra a l'utilisateur de retourner a l'instant precedant et de visualiser l'etat precedent de l'environnement, qu'il s'agisse des variables ou des poignees

`public swing.JButton fastforward` : Lorsque le bouton rewind a ete appuye au moins une fois, le JButton fastforward s'activera. En cliquant sur le JButton fastforward, l'utilisateur pourra avancer d'un cran dans le temps visualiser l'etat suivant de l'environnement, qu'il s'agisse des variables ou des poignees. Tant que l'environnement ne sera pas dans son etat courant, l'utilisateur ne pourra effectuer aucune operation sur les variables de l'environnement.

`public static Environnement quarantine` : l'environnement courant sera stocke dans l'attribut quarantine lors d'un retour en arriere afin de pouvoir le recuperer lorsque l'utilisateur retourne a l'etat courant et qu'il puisse poursuivre ses operations sur les variables de l'environnement courant

`public static boolean etatCourant` : etatCourant est a true si et seulement si l'environnement affiche par l'attribut paintbis est l'environnement courant. Des que l'utilisateur retourne en arriere dans le temps, etatCourant passe a false

`public swing.JList codeGenere` : JList qui affichera toutes les commandes Java generees par les operations de l'utilisateur

`public DefaultListModel commandesDeclenchees` : commandesDeclenchees contiendra les chaines de caractere correspondant aux commandes Java que les operations de l'utilisateur engendrent

`public static ChoixArguments choose` : l'attribut choose contiendra la fenetre qui s'affichera lorsque l'utilisateur devra choisir les valeurs des arguments d'un constructeur ou d'une methode. Il s'agira d'une instance de la classe ChoixArguments

`public swing.JButton create` : lorsque le constructeur a ete selectionne dans la JComboBox constructeurs, le constructeur selectionne sera appele en cliquant sur le JButton create

`public static JButton modifyattribute` : Lorsqu'une variable a ete selectionnee dans la JComboBox variables, que l'un de ses attributs a ete selectionne dans la JComboBox attributs et qu'une variable de type compatible avec l'attribut choisi a ete selectionnee dans nouveauxattributsnom, l'attribut selectionne de la variable choisie est affecte a sa nouvelle valeur choisie dans la JComboBox nouveauxattributsnom en cliquant sur le JButton modifyattribute. Lorsqu'une poignee a ete selectionnee dans la JComboBox variables et qu'une poignee ou variable de type compatible a ete selectionnee dans la JComboBox nouveauxattributsnom, la poignee choisie est affectee a sa nouvelle valeur choisie dans nouveauxattributsnom en cliquant sur le JButton modifyattribute.

`public static JButton ramassemiette` : le ramasse-miettes sera appele en cliquant sur le JButton ramassemiette qui lance la methode ramasseMiettes sur l'environnement courant stocke dans envCourant

`public JButton callmethod` : Lorsqu'une methode a ete selectionnee dans la JComboBox methodes et que la variable sur laquelle la methode sera appelee a ete selectionnee dans la JComboBox devant-methode, la methode choisie sera appelee sur la variable choisie en cliquant sur le JButton callmethod. Si la methode requiert des arguments, une instance de ChoixArguments sera creee et stockee dans l'attribut choose de GUI afin que la fenetre permettant a l'utilisateur de choisir la valeur de chaque argument apparaisse.

4.4.2 Methods

`public static void refreshPoigneesEtVariables()` : Lorsqu'une poignee a ete selectionnee dans la JComboBox variables, les variables et poignees de type compatible sont rajoutees dans la JComboBox nouveauxattributsnom refreshVariables

`public static void refreshVariables()` : vide la JComboBox variables de son contenu et y rajoute les noms de toutes les variables et poignees de l'environnement courant

`public static void refreshAttributs()` : vide la JComboBox attributs de son contenu et rajoute les noms des attributs de la variable selectionnee dans la JComboBox variables. Initialise ensuite la JComboBox nouveauxattributsnom avec les variables de type compatible avec le premier attribut rajoute dans la JComboBox attributs

`public static void refreshNouveauxattributsnom()` : rajoute dans la JComboBox nouveauxattributsnom les variables de type compatible avec l'attribut selectionne dans la JComboBox attributs de la variable selectionnee dans la JComboBox variables. Si aucune variable compatible n'existe, le JButton modifyattribute sera desactive. Sinon, il sera active.

`public static void initialiseTypeEtTypemethodes()` : rajoute dans la JComboBox types tous les types que l'utilisateur pourra creer et initialise les JComboBox constructeurs et methodes avec les constructeurs et methodes du premier type en appelant les methodes refreshConstructeurs et refreshMethodes

`public static void refreshConstructeurs()` : Lorsqu'une classe a ete selectionnee dans la JComboBox type, les noms simples des constructeurs declares dans cette classe seront stockes dans l'attribut constructeurs

`public static void refreshMethodes()` : Lorsqu'une classe a ete selectionnee dans la JComboBox type, les noms simples des methodes declarees dans cette classe seront stockees dans la JComboBox methodes

`public static void refreshDevantmethode()` : Lorsqu'une classe a ete selectionne dans la JComboBox type, les noms des variables de l'environnement sur lesquelles les methodes declarees dans la classe choisie peuvent etre appelees seront stockees dans la JComboBox devantmethode

`public static void refreshComboBoxes()` : remet a jour les JComboBoxes variables, devantmethode et variables

`public static void cloneMemoryState(String nouvelleLigneCode)` : copie les variables et les poignees de l'environnement courant. Ces copies sont respectivement ajoutees aux attributs instanceRecordingTape et pileRecordingTape de GUI. La chaine de caractere passee en argument est ajoutee a l'attribut commandesDeclenchees et est ensuite selectionnee

`public static void deactivateButtons()` : desactive tous les boutons de l'interface

`public static void activateButtons()` : active tous les boutons de l'interface

4.5 Poignee.java

Poignee de l'environnement pointant vers une variable de l'environnement. Lors de sa creation, la poignee aura pour type declare le type de la premiere variable a laquelle elle sera affectee. Cependant, la poignee pourra ulterieurement etre affectee a une variable n'ayant pas le meme type que le type declare de la poignee mais un type compatible, ie un type heritant du type declare initialement.

4.5.1 Fields

`public Variable pointeVers` : variable vers laquelle la poignee pointe

`public String nom` : nom de la poignee dans l'environnement

`public Class typepgn` : type declare de la poignee lors de sa creation

`public int caseMemoire` : entier indiquant le numero de la case memoire de la poignee

`public String idPoignee` : identifiant du Vertex JGraphX representant la poignee dans le graphe affiche par DrawEnvironnementBis

4.5.2 Method

`public Poignee clonePoignee()` : renvoie une copie d'une poignee de l'environnement. La copie de sera pas affectee par les modifications faites sur la poignee originelle

4.6 RetourMethode.java

fenetre permettant a l'utilisateur d'affecter le retour d'une methode : - soit a une nouvelle poignee qui sera creee dans l'environnement - soit a une poignee de type compatible qui existe deja dans l'environnement - soit a un attribut de type compatible d'une variable existant deja dans l'environnement

4.6.1 Field

`public static JFrame retourMeth` : JFrame qui s'affichera lorsque l'utilisateur devra choisir comment affecter un retour non void d'une methode

`public static Object valeurRetour` : valeurRetour contient le retour non void de la methode

`public static Variable variableRetour` : Si le retour de la methode est deja present dans l'environnement, variableRetour contiendra la variable de l'environnement dans laquelle ce retour est stocke. Si le retour de la methode n'existe pas dans l'environnement, une nouvelle variable dans laquelle sera stocke ce retour sera creee et ajoutee a l'environnement. Dans ce cas, variableRetour contiendra cette variable nouvellement creee.

`public static JButton newpoignee` : le JButton newpoignee affecte le retour de la methode a une nouvelle poignee qui sera creee et ajoutee a l'environnement

`public static String nomsarguments` : nomsarguments sera une chaine de caractere specifiant les noms de chaque argument qui fut passe a la methode qui a ete appelee

`public static JComboBox<String> varscompatibles` : les variables possedant au moins un attribut de type compatible avec le retour de la methode seront stockes dans la JComboBox varscompatibles

`public static JComboBox<String> attributscompatibles` : les noms des attributs de la variable selectionnee dans la JComboBox varscompatibles ayant un type compatible avec le retour non void de la methode seront stockes dans la JComboBox attributscomaptibles

`public static ArrayList<Field> attributscompatiblesvals` : les attributs de la variable selectionnee dans la JComboBox varscompatibles ayant un type compatible avec le retour non void de la methode seront stockes dans la JComboBox attributscomaptibles

`public static JButton affecteval` : lorsqu'une variable et son attribut de type compatible avec le retour non void de la methode ont ete selectionnes respectivement dans les JComboBox varscompatibles et attributscompatibles, le retour sera affecte a l'attribut selectionne de la variable selectionnee en cliquant sur le JButton affecteval

`public static JComboBox<String> poigneesCompatibles` : les noms des poignees de type compatible avec le retour non void de la methode seront stockes dans la JComboBox poigneesCompatibles

`public static JButton affectePoignee` : lorsqu'une poignee de type compatible avec le retour non void de la methode a ete selectionnee dans la JComboBox poigneesCompatibles, le retour de la methode sera affecte a la poignee en cliquant sur le JButton affectePoignee

4.6.2 Method

`public static void refreshPoigneesCompatibles()` : les noms des poignees de type compatible avec le retour non void de la methode sont ajoutees a la JComboBox poigneesCompatibles

`public static void refreshAttributsCompatibles()` : les noms des attributs de la variable selectionnee dans la JComboBox varscompatibles ayant un type compatible avec le retour de la methode sont stockes dans la JComboBox attributscompatibles et l'attribut lui-meme sera stocke dans l'ArrayList attributscompatiblesvals

4.7 Variable.java

Variable de l'environnement contenant une instance d'une classe Java

4.7.1 Field

`public Class type` : type declare de la variable lors de sa creation

`public String name` : nom de la variable dans l'environnement

`public Object valeur` : valeur de l'instance Java cree. Comme l'application est completement generique, toutes les valeurs seront de type Object

`public ArrayList<Poignee> poignees` : liste des poignees pointant vers la variable

`public ArrayList<Variable> variablesMeres` : Une variable mere peut avoir plusieurs attributs de meme type et de meme valeur, ie pointant vers la meme variable fille. Pour les differencier il faut donc stocker dans la variable fille chaque variable mere mais aussi le nom de l'attribut de la variable mere. A tout moment, variablesMeres et attributsPeres seront de meme taille

`public ArrayList<String> attributsPeres` : nom de l'attribut pere ayant pour valeur la variable

`public ArrayList<String> identifiantsPeres` : identifiants des noeuds representant les attributs peres dans le graphe affiche par DrawEnvironnementBis

`public String identifiant` : identifiant du noeud representant la variable dans le graphe affiche par DrawEnvironnementBis

4.7.2 Method

`public void retirePoignee(String namepgn)` : retire la poignee ayant pour nom namepgn de la liste de poignee pointant vers la variable

`public Variable cloneVariable()` : Copie une variable ainsi que sa valeur. La copie d'une variable ne sera donc pas affectee par les methodes appelees sur la variable originelle ni par les modifications d'attributs de la variable originelle

`public boolean possedeAttributDans(Variable varAtt)` : renvoie true si et seulement si au moins l'un des attributs de la variable a pour valeur varAtt

5 Instantiation and Field Modification Application Javadoc

5.1 DrawDiagram.java

Diagramme dans lequel chaque noeud contiendra le nom des variables de l'environnement ainsi que les valeurs de leurs attributs

5.1.1 Fields

`public static Graphics2D graph2d` : graphe sur lequel les noeuds representant toutes les instances seront affichees

`public static int nbColumns` : nombre de noeuds affiches par ligne

`public static int sizeRectX` : longueur d'un noeud representant une variable de l'environnement

`public static int sizeRectY` : largeur d'un noeud representant

5.1.2 Methods

`public void paint(Graphics g)` : le diagramme est trace en appelant paint ou repaint sur une instance de DrawDiagram

`private void drawDiag()` : affiche un noeud pour chaque variable de l'environnement qui contiendra son nom et la valeur de tous ses attributs

5.2 DrawInstance.java

Graphe sur lequel le dessin representant une instance de la classe racine de GUI sera representee. La representation de chaque instance dependra des valeurs de ses attributs.

5.2.1 Fields

`public java.awt.Graphics2D inst` : graphe sur lequel le dessin representant une instance sera representee

`public Variable varInitiale` : variable qui sera representee sur un dessin

5.2.2 Method Detail

`public void drawInst(Variable var)` : affiche un dessin representant la variable var sur l'attribut inst de DrawInstance

`public static Method getMethodByName(String methodName, Class type)` : recupere une methode d'une classe a partir de son nom methodName

`public static boolean hasEnumFields(Class type)` : renvoie true si et seulement si la classe possede au moins un attribut de type Enum

`public static boolean estPrimitif(Class type)` : renvoie true si et seulement si la classe est primitive ou est de type String

5.3 GUI.java

Classe contenant l'interface de l'application Instanciation et Modification d'attributs

5.3.1 Fields

`public static JFrame menu` : fenetre qui apparaitra lorsque l'application sera executee

`public static Class racine` : classe principale qui sera d'abord instanciee avant de creer tous ses attributs ayant une valeur null

`public static ArrayList<Variable> instances` : les variables creees dans l'environnement seront stockees dans l'attribut instances

`public static JButton create` : lorsque la classe principale racine ou l'attribut d'une variable ayant une valeur null a ete selectionne dans la JComboBox chooseCreate, cette variable ou attribut sera cree en appuyant sur le JButton create

`public static JButton modify` : lorsqu'une variable a ete selectionnee dans la JComboBox chooseModify, les attributs de type Enum seront modifies en appuyant sur le JButton modify

`public static int nbInstancesMax` : nombre maximal d'instances de type racine affichees possible. Ce nombre sera egal a nbColonnes*nbLignes

`public static int nbColonnes` : nombre d'instances qui seront affichees sur une meme ligne

`public static int nbLignes` : nombre d'instances qui seront affichees sur une meme colonne

`public static JComboBox<String> chooseCreate` : Lorsque nbInstancesMax ¿ 0 , chooseCreate contiendra le nom simple de la classe racine. De plus, chooseCreate contiendra les noms des attributs des variables de l'environnement ayant la valeur null

`public static ArrayList<Field> chooseField` : Soit i l'indice de l'item selectionnee dans la JComboBox chooseCreate. lorsque l'attribut d'une variable ayant une valeur null est selectionne dans chooseCreate, la variable ayant cet attribut sera chooseField.get(i)

`public static java.util.ArrayList<Variable> chooseVar` : Soit i l'indice de l'item selectionnee dans la JComboBox chooseCreate. lorsque l'attribut d'une variable ayant une valeur null est selectionne dans chooseCreate, l'instance de la classe Field correspondant a cet attribut sera chooseVar.get(i)

`public static JComboBox<String> chooseModify` : les noms des instances creees possedant des attributs de type enum seront stockees dans la JComboBox chooseModify

`public static Modification modifyWindow` : lorsque l'utilisateur cliquera sur le JButton modify, une instance de Modification sera creee et stockee dans l'attribut modifyWindow afin que la fenetre permettant a l'utilisateur de selectionner la nouvelle valeur des attributs de type enum de la variable selectionnee dans chooseModify apparaisse

`public static ArrayList<DrawInstance> dessin` : les dessins representant chaque instance de type racine seront stockes dans l'attribut dessin, chacun etant une instance de DrawInstance

`public static JPanel ensembleDessins` : l'ensemble des dessins stockes dans l'attribut dessin seront affiches sur le JPanel ensembleDessins afin que l'utilisateur puisse les voir simultanement. Ils seront affiches sur le cote gauche de la JFrame stockee dans l'attribut menu

`public static DrawDiagram diagramme` : chaque instance creee sera representee dans un diagramme avec les valeurs de ses attributs. Ce diagramme sera ensuite affiche sur le cote droit de la JFrame stockee

dans l'attribut menu.

5.3.2 Method

`public static void addFieldsToInstances(Variable newvar)` : ajoute les attributs de valeur non null et de type non enum d'une instance a l'environnement. La fonction est ensuite appelee recursivement a chaque attribut qui a ete ajoute a l'environnement.

`public static int nbSameClass(Class classe)` : renvoie le nombre d'instances dans l'environnement ayant pour type la classe fournie en argument

`public static Variable getVariableByName(String nomVariable)` : renvoie la variable de l'environnement nommee nomVariable

`public static Field[] getAllFields(Class type)` : renvoie tous les attributs de la classe fournie en argument ainsi que tous les attributs herites des supertypes de cette classe

5.4 Modification.java

Fenêtre permettant a l'utilisateur de modifier les valeurs des attributs de type Enum d'une variable de l'environnement

5.4.1 Field

`public static JFrame window` : fenetre s'affichant lorsque l'utilisateur voudra changer la valeur des attributs enum d'une instance creee dans l'environnement

`public static Variable varModifiable` : la variable dont les attributs seront modifies

`public static ArrayList<DefaultListModel> listModel` : Ensemble de listes. Chaque liste contiendra toutes les valeurs possibles des attributs enum de la variable varModifiable

`public static ArrayList<JList> jList` : Ensemble de listes. Chaque liste contiendra toutes les valeurs possibles des attributs enum de la variable varModifiable

`public static ArrayList<JScrollPane> scrollPane` : Ensemble de listes. Chaque liste contiendra toutes les valeurs possibles des attributs enum de la variable varModifiable

`public static ArrayList<Object[]> valeursEnum` : nouvelles valeurs choisies par l'utilisateur des attributs de type Enum de la variable varModifiable

`public static ArrayList<Field> attributsEnum` : attributs de type Enum de la variable varModifiable

`public static JButton modifier` : Lorsque l'utilisateur aura selectionne les nouvelles valeurs de chaque attribut de type Enum de la variable varModifier, ces nouvelles valeurs seront affectees aux attributs en cliquant sur le JButton modifier

5.5 Variable.java

Variable de l'environnement contenant une instance Java et un nom

5.5.1 Field Detail

`public Object valeur` : instance Java stockee dans la variable

`public String nom` : nom de la variable dans l'environnement