Paris, le 18.12.2012

IKHLEF Rafik , 2303887.
étudiant n°20

Rapport Final Li314

L'objectif et de concevoir et de programmer en java une application qui met en œuvre un algorithme évolutionniste.

Ces algorithmes se dotent d'une population d'individus qui représente un ensemble de solution possible à un problème donné. Ce problème est représenté par un environnement.

-Les individus disposent d'une fitness qui mesure l'adéquation de leur comportement au probleme qui leur est posé par leur environnement.

-la population d'individus évolue de génération en génération.

Lors du passage d'une génération à la suivante, on élimine un certain nombre d'agents dont la *fitness* est faible et on les remplace par des nouveaux individus qui sont les descendants d'individus généralement plus performants. Au fil des générations, la *fitness* moyenne des individus de la population augmente globalement, si bien que les individus résolvent de mieux en mieux le problème.

dans le tme1 le package pobj.algogen contient classe Population, Individu, PopulationFactory et la classe PopulationMain.

La méthode de static createRandompopulation(int n) de la PopulationFactory créée une population de n d'individus généré aléatoirement (la liste d'individus) , chaque individu est représenté par sa valeur propre et sa fitness.

On voit bien que cela ressemble a une Factory statique ou (simple Factory), car sans la factory, le client(PopulationMain) doit créé une instance de Population (new Population()), mais dans le cas de Factory simple, le client se contente de faire : PopulationFactory.createRandomPopulation(n) ;.

voici un exemple d'une population de 10 individus.

```
le nombre d'individu à créer est : 10
la population aléatoire est : [[ 47.0, 0.0], [ 71.0, 0.0], [ 22.0, 0.0], [ 72.0, 0.0],[ 9.0, 0.0], [ 19.0, 0.0], [ 22.0, 0.0],
[ 1.0, 0.0], [ 85.0, 0.0], [ 74.0, 0.0]]
```

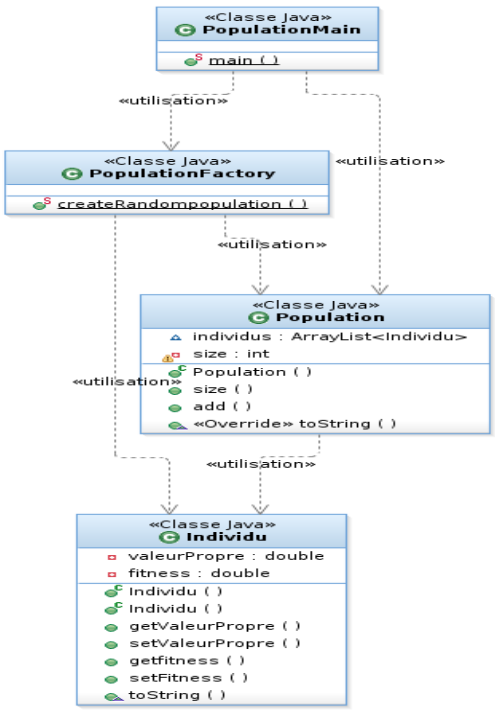


Diagramme Tme1

TME2 :

-dans le tme2 on souhaite implémenter les méthodes nécessaires a l'évaluation d'une population dans un environnement dont le but est d'identifier les meilleurs individus et de ne garder que ceux ci, pour engendrer une nouvelle génération. C'est l'environnement qui conditionne l'évaluation de la fitness des individus.

-pour cela on créé une interface Environnement , avec la méthode public double eval(Individu i) qui rend la fitness de l'individu i dans l'environnement courant.

-on définie la classe ValeurCible qui implémente l'interface Environnement, dans cette classe en définie le corps de la méthode public double eval(Individu i) et rajoute d'autres méthodes (accesseur, modificateur, deux constructeurs, et la toString()).

-dans la classe Population on définie la méthode public Population évaluer(Environnement e), cette derniere utilise la méthode eval de l'environnement pour évaluer la fitness de chaque individu de la population, et à l'aide des méthodes de la classe Collection on obtient une liste triée par ordre croissant de fitness.

-comme on compare des objets d'Individu, alors la classe Individu doit implémenter l'interface Comparable, et on redéfini la méthode compareTo(Individu o).

-toujours dans la classe individu, on rajoute d'autres méthodes comme la méthode de mutation, la méthode de croisement, et de clonage.

-pour conclure, dans la classe Population, on dote cette dernière de deux méthodes supplémentaires la méthode de mutation, reproduction et d'évolution de la population.

Voici un résultat d'évolution d'une population de 10 individus :

```
-------------------------TME2-----------------------------------------
la population aleatoire est : [[ 95.0, 0.0], [ 25.0, 0.0], [ 82.0, 0.0], [ 46.0, 0.0], [ 38.0, 0.0], [ 98.0, 0.0], [ 15.0, 0.0],
[ 76.0, 0.0], [ 33.0, 0.0], [ 22.0, 0.0]]

--évaluation  + tri décroissant + évolution (10 générations) d'une population crée avec une ArrayList--------------
[[ 15.0, 0.0], [ 15.75, 0.0], [ 16.14375, 0.0045725637248375 27], [ 16.5375, 0.004141803080199466], [ 16.14375,
0.004141803080199466], [ 16.5375, 0.004141803080199466], [ 16.5375, 0.004141803080199466], [ 16.14375, 0.004141803080199466],
[ 15.75, 0.0045725637248375 27], [ 16.14375, 0.004141803080199466]]
```

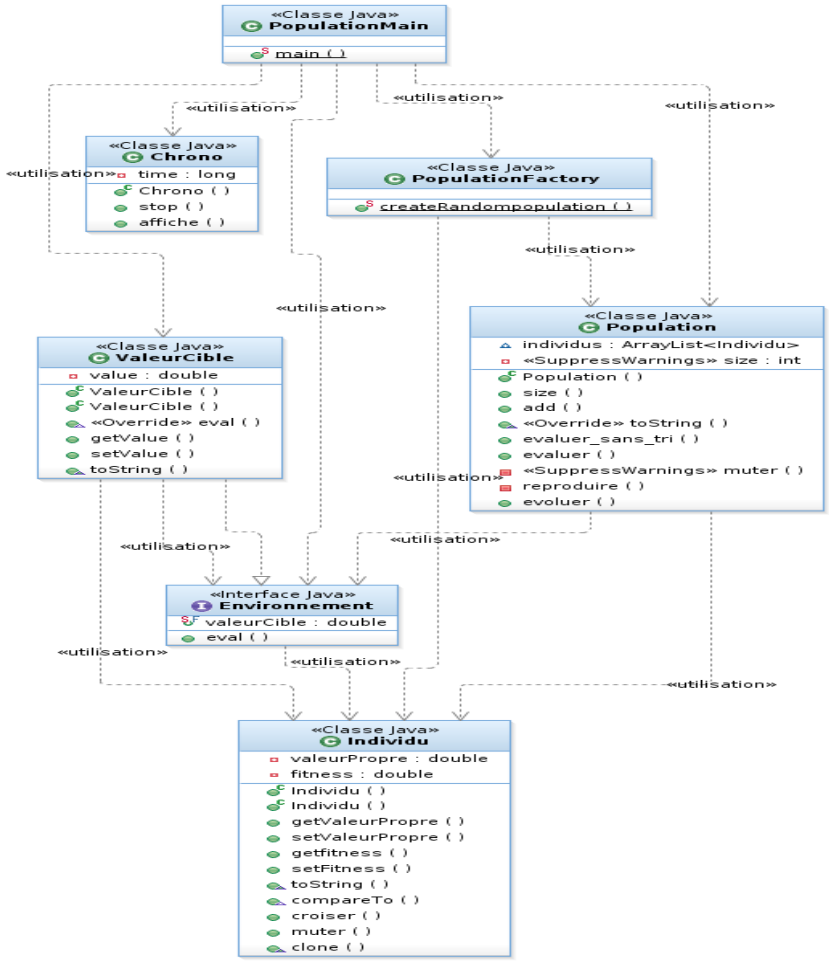


Diagramme Tme2

Tme3

-l'objectif du Tme3 est de généré des expressions arithmétiques et de pouvoir les calculées. Dans cette partie du projet on souhaite utilisé le DP composite dont le composant est une expression, il est composé des constantes, des variables et des opérateurs. Chaque expression est évaluée dans un environnement qui affecte des valeurs à ces variables.

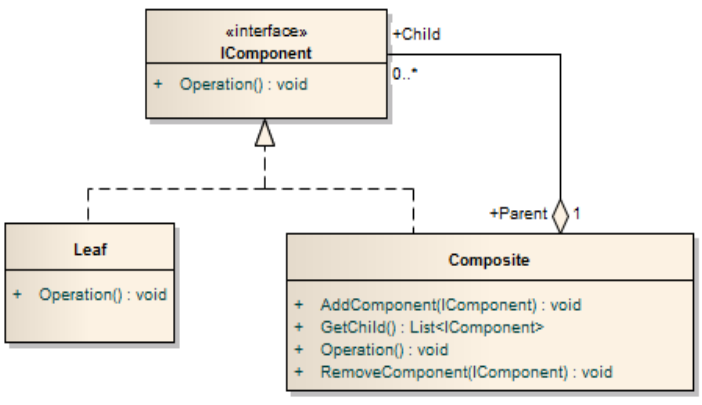
-comme vu précédemment(tme1), on souhaite utilisé une simple factory static, pour que le client ne crée pas lui même des objet, pour cela on utilise une factory.

le DP Composite :

le but du pattern Composite est d'offrir un cadre de conception d'une composition d'objets dont la profondeur est variable, cette conception étant basée sur un arbre. Cette composition est encapsulée vis-à-vis des clients des objets qui peuvent interagir sans devoir connaître la profondeur de la composition.

Le pattern composite résout le problème des expressions en utilisent la composition récursive. Cette composition récursive est nécessaire car une expression peut posséder des expressions (left , right) qui possèdent elles mêmes d'autres expressions.

- voici une modèle du DP Composite :



- l'interface Icomponent possède une seule méthode nommée opération() qui sera implémentée dans les sous classes Leaf et Composite.

- l'interface Expression (Icomponent) : elle possède 3 sous classes concrètes à savoir Constante, Variable et OperateurBinaire , cette dernière détenant une association d'agrégation avec l'interface Expression représentant les liens avec ses expressions. dans notre expression on trouve la méthode eval(EnvEval), elle prend en paramètre un environnement dont le quel l'expression est évaluée.

- classe Constante (leaf) : implémente l'interface Expression, donc c'est une expression, on défini le corps de la méthode eval, et on d'autres méthodes spécifique à la classe constante comme la méthode getValue (renvoie la la valeur de la constante) , la méthode toString(renvoie la chaîne de caractère de la constante) et un constructeur de constante.
- Classe Variable(leaf) : implémente aussi l'interface Expression, donc c'est une expression on défini aussi le corps de la méthode eval, et d'autres méthodes spécifique à la classe Variable comme le constructeur Variable et la méthode toString .
- Classe OperationBinaire (composite) : implémente aussi l'interface Expression, donc c'est une expression, on défini aussi le corps de la méthode eval, et d'autres méthodes spécifique comme toString, getLeft(renvoie l'expression de gauche), getRight(renvoie l'expression de droite) et biensur le constructeur Operateurbinaire qui prend on paramètres un operator, une expression gauche et une expression droite.

-la classe EnvEval : est l'environnement dont le quel est évaluée notre expression arithmétique, elle est représentée par un tableau de double.

- la classe TestExpression (le client) : elle possède la méthode main, mais le ne travaille pas directement sur les expressions ou la composite, donc elle ne connais pas leurs existence. Elle travaille avec une classe qui possede des méthode static.

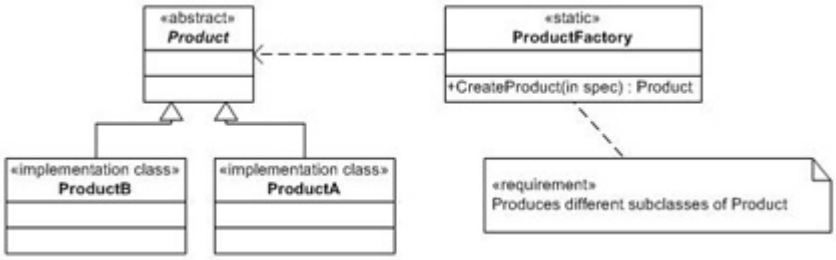
-la classe ExpressionFactory : elle possède différentes méthodes static avec les quelles le client crée les différentes expressions. Les differentes méthodes sont :

```
public static Expression createOperateurbinaire(Operator op,Expression ex,Expression exp).
public static Expression createConstant(double constant)
public static Expression createVariable(int id)
public static Expression createRandomExpression()
public static Expression createRandomExpression(int profondeur)
public static EnvEval createRandomEnvironment()
```

c'est a l'aide de ces méthodes static que le client crée les expressions et l'environnement d'évaluation.

Voici un schéma du DP Factory, sauf que dans notre cas (tme3) on travaille sur des expressions et dans le cas du tme1 et tme2 on travaille directement sur une classe concrète qui est la classe Population pour crée la population d'individus , c'est à dire sans passer par une classe abstraite.

DP simple Factory (non static)



voici quelques résultats de test du tme3 :

```
------------teste de la classe Expressionfactory ------
exemple d'un environnement aleatoire : [0.0540174277258092, 0.16599481175334552]
**************************************************
evoluer une expression binaire simple :
(X0 + X1) = 20.0
**************************************************
evaluer une expression binaire avec profondeur :
[0.12210097834896971, 0.917525816292196]
((X1 * (X1 * pobj.arith.Constante@2ada52a1)) * pobj.arith.Constante@6366de01) = 0.3942830340183895
```

voici le diagramme des classes du tme 3 :

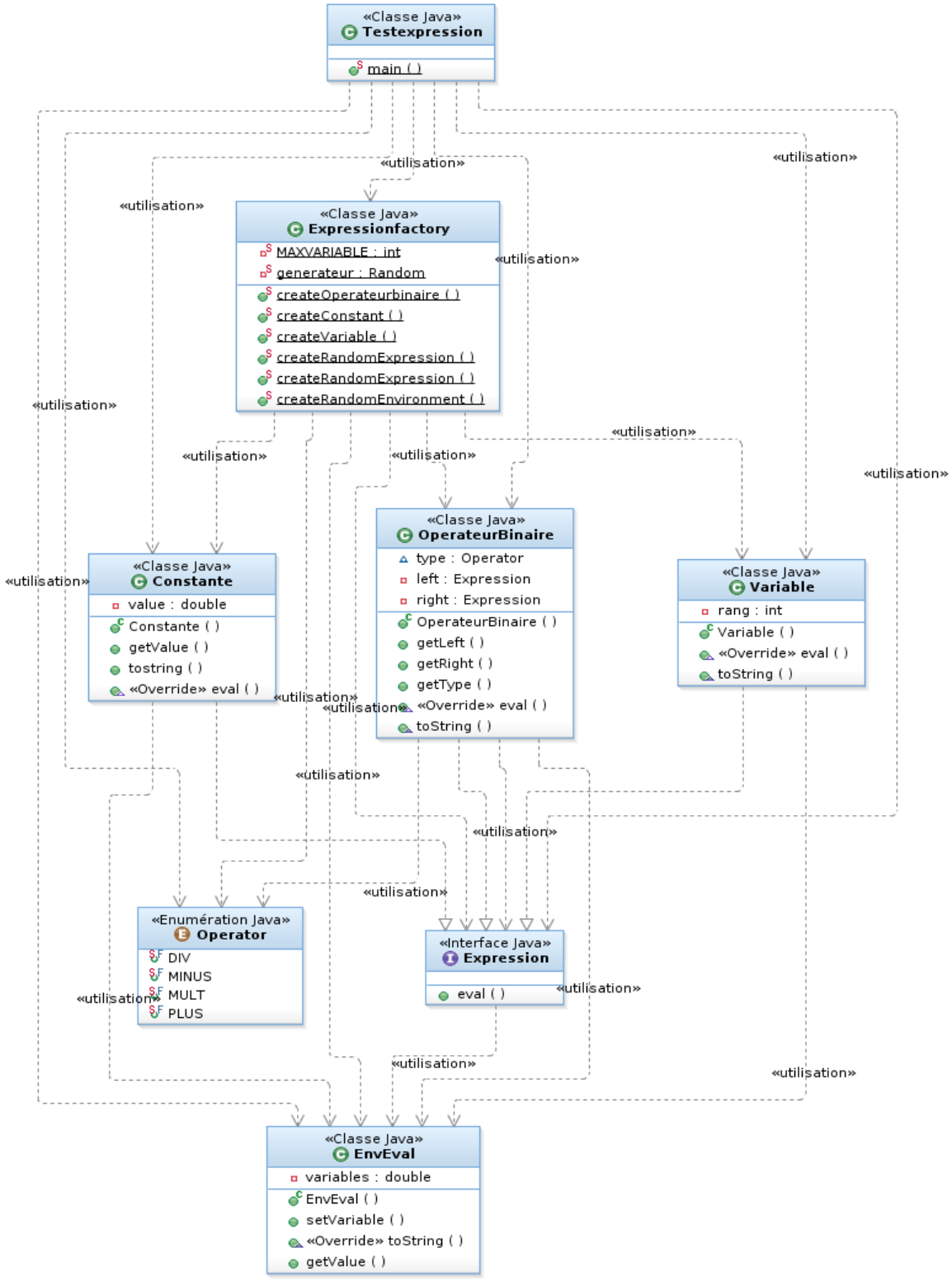


Diagramme Tme3

TME4

-dans cette partie du tme, on fusionne le tme2 et le tme3. On va faire en sorte que les expressions soient manipulés au sein de l'algo génétique du tme2. C'est a dire évaluer une population d'expression.

- vu qu'on souhaite s'appuyer sur de la délégation pour ne pas modifier les expressions du tme3, et comme on comme on souhaite s'appuyer sur l'interface Expression, alors la délégation qui consiste de prendre en paramètre une Expression dans la classe individu.

La valeur propre de l'individu devient une expression, on implémente les méthodes croiser et muter de l'individu. Toujours dans la classe individu, comme on a besoins de comparer des objets individu (les expressions) selon leurs fitness, dans ce cas , la classe individu implémente l'interface Comparable , et on défini la méthode compareTo.

-pour évaluer les expressions, on implémente **public double** eval(Individu i)dans la classe ValeurCible, cette fonction renvoie le fitness de l'expression.

- on ajoute un EnvEval qui donne la valeurs des variables et une valeur qui donne la valeur cible de la fonction dans l'environnement.
- pour éviter une explosion de la taille des expressions, on doit les simplifier, pour cela on ajoute la fonction **public** Expression simplifier() à l'interface expression et ses sous classes.
- Toujours dans l'interface expression on ajoute la fonction **public** Expression clone(), dans le but qu'il n'y ait pas de dépendances entre l'expression d'origine et sa version simplifiée, pour ne pas modifier le parent d'une expression fils, donc pour ne pas modifier la population. Ce que donne des expressions immuables.

Exemple de test du tme4 , pouplation de taille 20, sur 10 générations

```
------------------------------population avec tri decroissant-------------------------------------
la population des individus [valeurPropre, fitness] apres evaluation,tri decroissant par fitness ,  :
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0,
2.69202015445041]]

------------------------------population evoluer, generation  : 1---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0,
2.69202015445041]]
------------------------------population evoluer, generation  : 2---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041],
[ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041]]
------------------------------population evoluer, generation  : 3---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X0, 2.69202015445041],
[ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041]]
------------------------------population evoluer, generation  : 4---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [
X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041]]
------------------------------population evoluer, generation  : 5---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0,
2.69202015445041]]
------------------------------population evoluer, generation  : 6---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X0, 2.69202015445041],
[ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041]]
------------------------------population evoluer, generation  : 7---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [
X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041]]
------------------------------population evoluer, generation  : 8---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0,
2.69202015445041]]
------------------------------population evoluer, generation  : 9---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X0, 2.69202015445041],
[ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041]]
------------------------------population evoluer, generation  : 10---------------------------------
[[ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1,
143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X1, 143.88085449515305], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [
X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041], [ X0, 2.69202015445041]]
```

voici le diagramme du package pobj.algogen du tme4 :

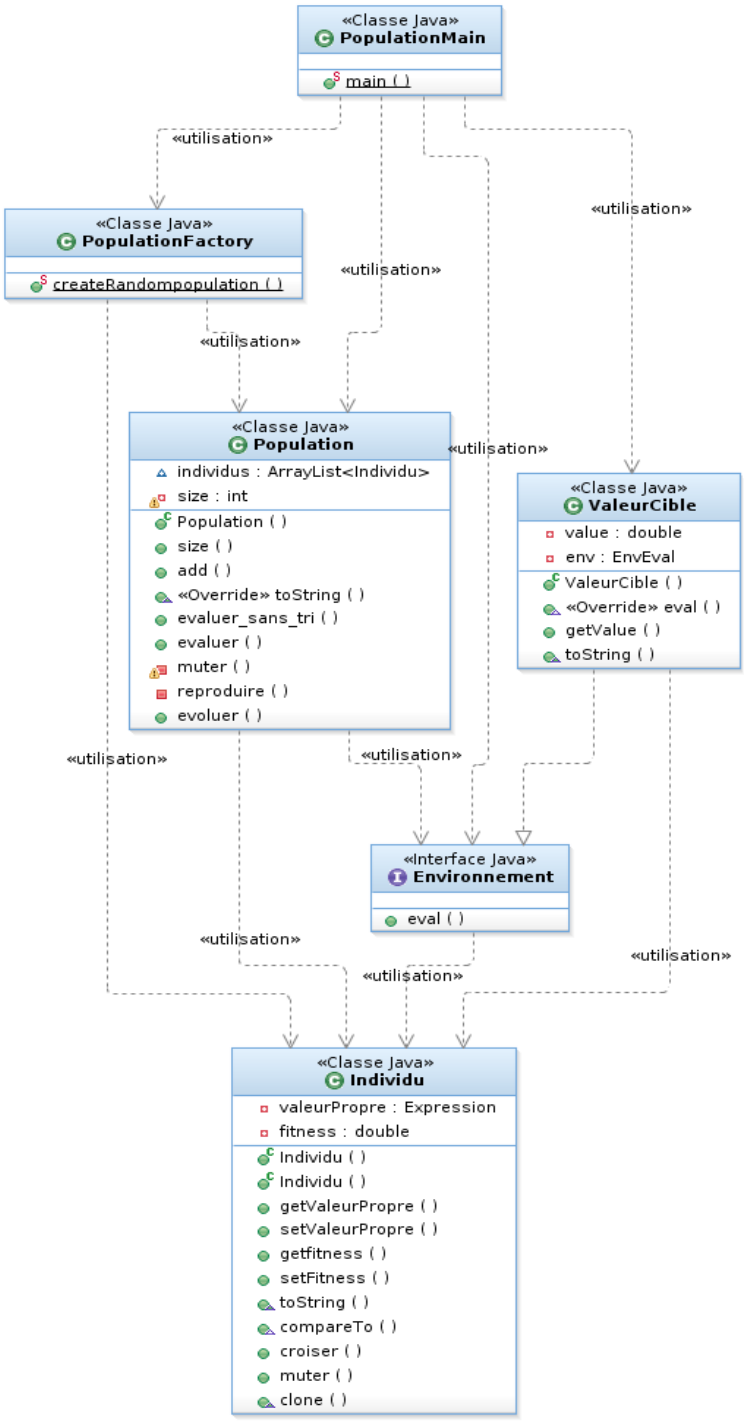


Diagramme Tme4_pobj.algogen1

voici le diagramme du package pobj.arith du tme4

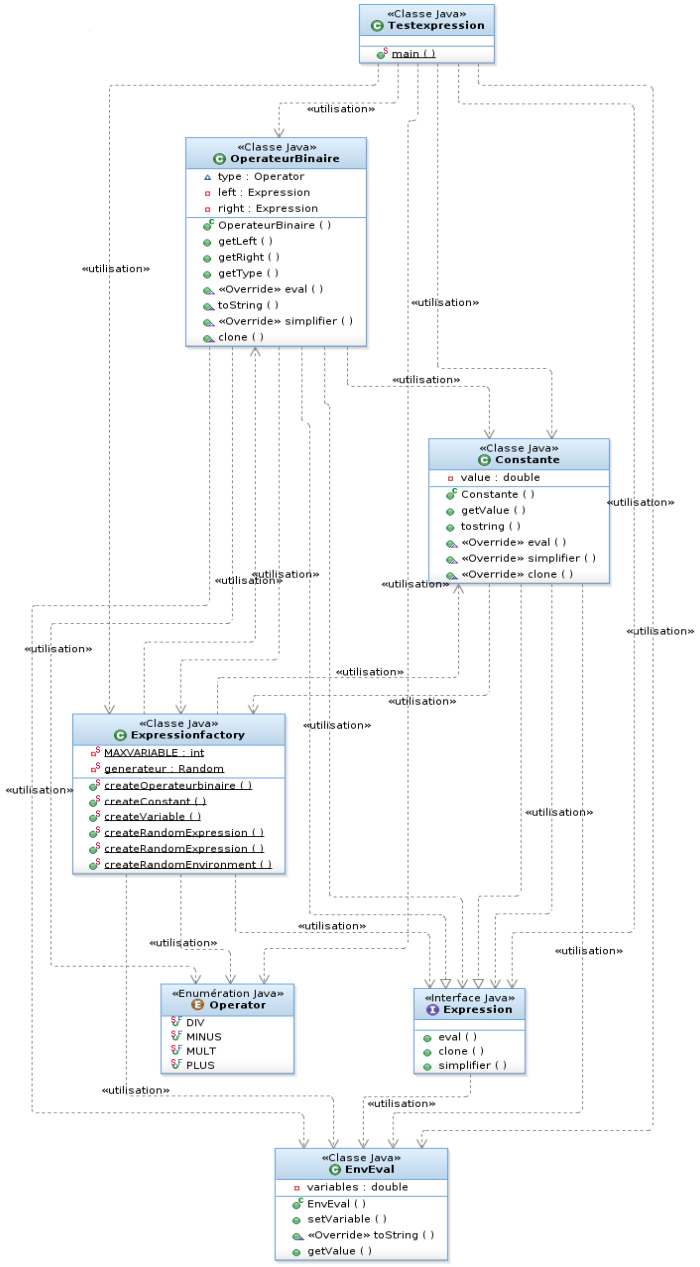


Diagramme Tme4 pobj.arith2

****

TME5

on souhaite étudier le déplacement d'un agent dans un labyrinthe, son déplacement est commandé par un contrôleur qui indique à l'agent comment ce déplacer dans le labyrinthe à partir de ce qu'il perçoit autour de lui.

Le labyrinthe doit être entouré d'une enceinte de mur pour que l'agent ne soit pas une case au bord, et la case du départ dans la qu'elle l'agent apparaît au départ doit être vide.

Pour s'assurer que le chargement d'un labyrinthe respecte les conditions, on a choisi d'utiliser les exceptions.

-Dans le package exception, j'ai définis la classe LabyErroneException qui hérite de exception , elle contient un constructeur qui prend en paramètre un point (le point a vérifié)et un message(la chaîne de caractère à retourner a l'aide du super(message) de la classe mère). `public class LabyErroneException extends Exception`

-la classe CaseDepartNonVideException hérite de LabyErroneException, contient un constructeur qui prend un point en paramètre et quit fait appel au constructeur de la super classe. `public class CaseDepartNonVideException extends LabyErroneException`

- la classe LabyMalEntoureException hérite de LabyErroneException, contient un constructeur qui prend en parametre un point et qui fait appel au constructeur de la super classe. `public class LabyMalEntoureException extends LabyErroneException`

les Exceptions : Les exceptions représentent le mécanisme de gestion des erreurs intégré au langage Java. Il se compose d'objets représentant les erreurs et d'un ensemble de trois mots clés qui permettent de détecter et de traiter ces erreurs (try, catch et finally) et de les lever ou les propager (throw et throws).

-Lors de la détection d'une erreur, un objet qui hérite de la classe Exception est créé (on dit qu'une exception est levée) et propagé à travers la pile d'exécution jusqu'à ce qu'il soit traité.

Si dans un bloc de code on fait appel à une méthode qui peut potentiellement générer une exception, on doit soit essayer de la récupérer avec try/catch, soit ajouter le mot clé throws dans la déclaration du bloc. Throws permet de déléguer la responsabilité des erreurs vers la méthode appelante.

-L'exécution totale du bloc try et d'un bloc d'une clause catch sont mutuellement exclusives : si une exception est levée, l'exécution du bloc try est arrêtée et si elle existe, la clause catch adéquate est exécutée.

-Pour générer une exception, il suffit d'utiliser le mot clé throw, suivi d'un objet dont la classe dérive de Throwable. Si l'on veut générer une exception dans une méthode avec throw, il faut l'indiquer dans la déclaration de la méthode, en utilisant le mot clé throws.

-Les méthodes pouvant lever des exceptions doivent inclure une clause throws nom_exception dans leur en-tête. L'objectif est double : avoir une valeur documentaire et préciser au compilateur que cette méthode pourra lever cette exception et que toute méthode qui l'appelle devra prendre en compte cette exception (traitement ou propagation).

- la simple factory, la classe VerificationLaby contient les contrôles des exceptions dans la méthode **public static void** verifierconditions(Labyrinthe l)**throws** LabyErroneException qui contient deux méthodes qui délègue le traitement des exceptions.

- la méthode **public static void** estCaseInitialevide(Labyrinthe l)**throws** CaseDepartNonVideException lève une exceception si la case depart est non vide.

- la méthode **public static void** estEntoureDeMurs(Labyrinthe l)**throws** LabyMalEntoureException lève une excepetion si les bord ne sont pas des murs.

- la méthode **public static int** corrigerLabyrinthe(Labyrinthe l) **throws** LabyErroneException corrige les erreurs détectées et renvoie le nombre d'erreurs détectées.

Dans le pacquage agent, j'ai crée la classe CorrectionErreur **public class** CorrectionErreur (pour des tests)qui corrige des labyrinthes qui sont soit mal entourée soit leurs case initial est vide.

-Junit : Le but est d'automatiser les tests. Ceux-ci sont exprimés dans des classes sous la forme de cas de tests avec leurs résultats attendus. JUnit exécute ces tests et les comparent avec ces résultats.
Cela permet de séparer le code de la classe, du code qui permet de la tester. Souvent pour tester une classe, il est facile de créer une méthode main() qui va contenir les traitements de tests. L'inconvénient est que ce code "superflu" est inclus dans la classe.

Dans le pacquage test.agent j ai défini la classe **public class** AgentTest **extends** TestCase, et dans la méthode **public void** testMesurePerf() des tests (assertTrue et asserFalse) qui permettent de vérifier le score réalisé par l'agent dans le labyrinthe.

Voici les différents résultats :

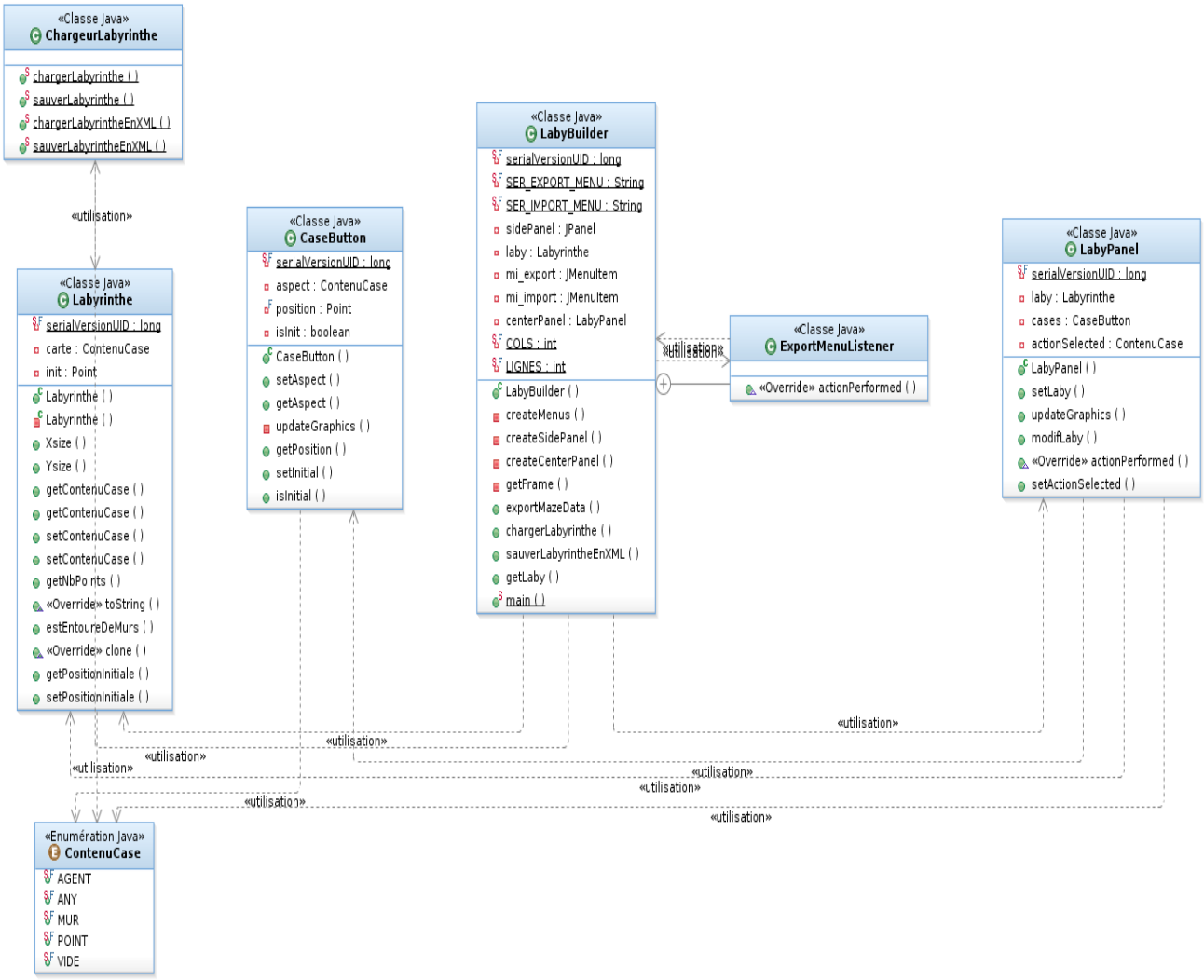
résultat de la classe : CorrectionErreur

```
exemple erreur pour le fichier foufi
java.awt.Point[x=1,y=1]
java.awt.Point[x=1,y=0]
java.awt.Point[x=2,y=0]
java.awt.Point[x=14,y=1]
le nbre d'erreur est : 4
exemple erreur pour le fichier foufi1
java.awt.Point[x=1,y=1]
java.awt.Point[x=3,y=0]
java.awt.Point[x=8,y=0]
java.awt.Point[x=14,y=4]
java.awt.Point[x=0,y=7]
java.awt.Point[x=0,y=8]
java.awt.Point[x=14,y=8]
le nbre d'erreur est : 7
```

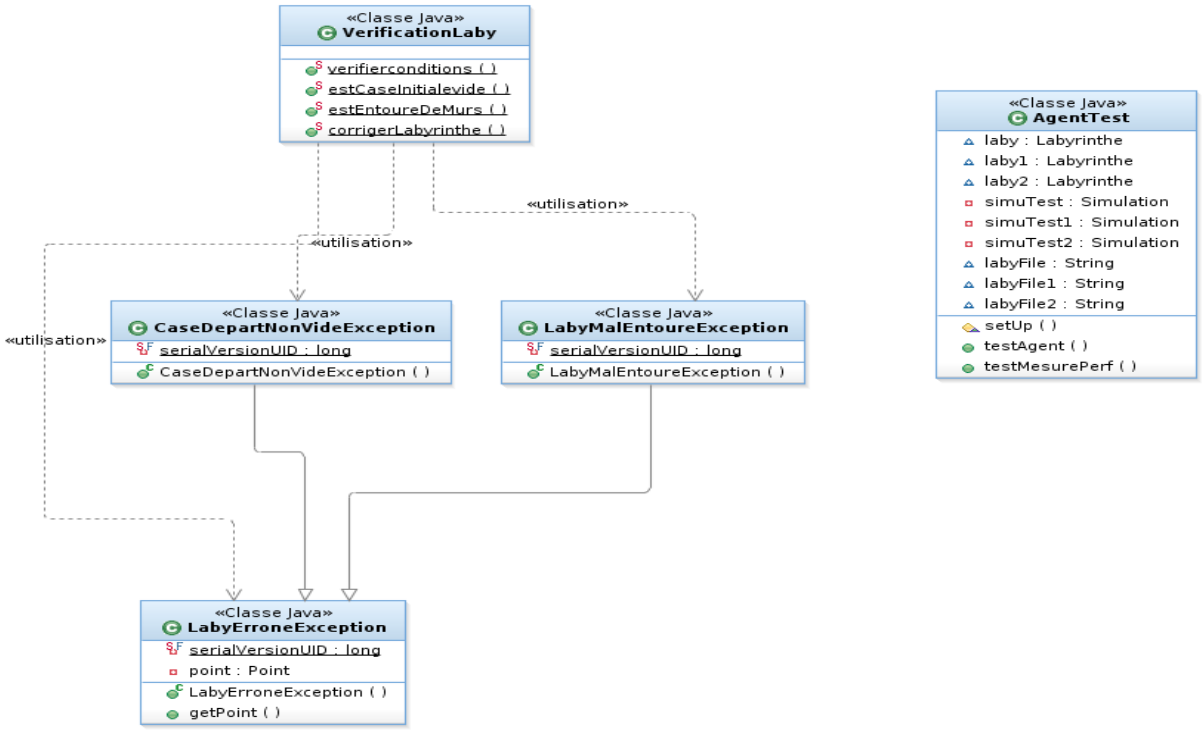
resultat de la classe AgentTest : tout les test sont de couleur vert

```
le score est : 12
le score est : 2
le score est : 2
```

voici le diagramme du pacquage agent.laby du tme5



voici Diagramme Tme5_Exception & agent_test :



TME6

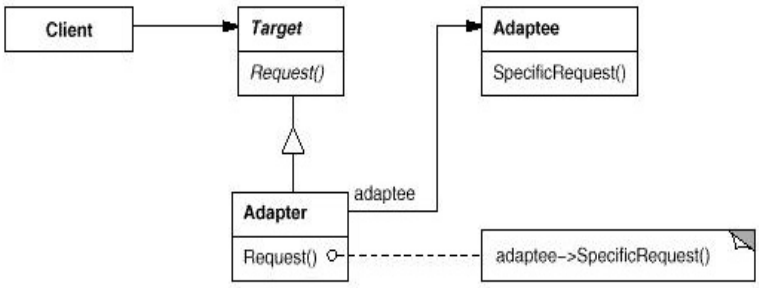
dans cette partie on souhaite réutiliser les classes d'évolution génétique pour rechercher par évolution un contrôleur d'agent qui maximise le nombre de cases visitées par l'agent. Pour cela on modifié le pacquage pobj.algogen de façon à pouvoir utiliser le même code de manipulation génétique pour manipuler un double, une expression ou un agent dans un labyrinthe(évolution d'une population). Pour cela on choisit un adapter.

Adapter : le but du pattern adapter est de convertir l'interface d'une classe existante en l'interface attendue par des clients également existants afin qu'ils puissent travailler ensemble. Il s'agit de conférer à une classe existante une nouvelle interface pour répondre aux besoins de clients.

Le dp adapter est utilisé dans les cas suivants :

- pour intégrer dans un système un objet dont l'interface ne correspond pas à l'interface requise au sein de ce système.
- pour fournir des interfaces multiples à un objet lors de sa conception.

Voici le diagramme du dp adapter :



le pattern adapter propose une solution qui consiste à créer les classes ControleurIndividuAdapter qui possède une association avec l'objet Icontroleur, et IndividuExpression qui possède une association avec l'objet Expression .les deux classes héritent de AbstarctIndividu, cette dernière implémente l'interface Individu. L'implantation des méthodes de l'interface Individu consiste à déléguer correctement les appels aux objets Expression et Icotroleur.

dans notre pacquage pobj.algogen on trouve :

-l'inreface Environnement implémentée par la classe ValeurCible et par la classe LabyEnvironnementAdapter.

-**l'interface Individu (Target)** : introduit la signature des méthodes de l'objet(croiser, muter et clone).

- la classe AbstractIndividu : implémente l'interface Comparable et a pour critère de comparaison la fitness. représente un individu abstrait et porte seulement la fitness de l'individu.
- la classe Population : implemente l'interface Ipopulation, et l'interface Iterable<Individu> et qui implante des méthodes manipulant la liste d'individus.
- dans le pacquage pobj.algogen.adapter.arith : - la classe **IndividuExpression(Adapter)** : hérite de AbstarctIndividu qui implémente Individu , implante les méthodes de l'interface Individu en invoquant les méthodes de l'objet adapté Expression.

On trouve aussi dans ce pacquage(pobj.algogen.adapter.arith) les classes:ArithPopulationMain, PopulationFactory et ValeurCible.

- dans le pacquage pobj.algogen.adapter.agent : - la classe LabyEnvironnementAdapter : est un Environnement, sa méthode eval permet de récupérer le fitness de l'individu suite a la simulation.
- la classe ContrôleurIndividuAdapter(Adapter) :hérite de AbstarctIndividu qui implémente Individu , implante les méthodes de l'interface Individu en invoquant les méthodes de l'objet adapté IControleur.

On trouve aussi dans ce pacquage (pobj.algogen.adapter.agent) les classes EvolutionAgentmain et PopulationFactroy.

-Les interfaces **Expression** (pacquage pobj.arith) et **Icontroleur** (pacquage agent.control) :(Adaptée) introduit l'objet dont l'interface doit etre adaptée pour correspondre à l'interface Individu.

Voici le résultat d'une execution de la classe EvulutionAgentMain d'une population de 10 indivdus :

évolution de la population : le meilleur individu
la meilleur Individu de la derniere generation est : [.??? ??? -> GAUCHE, ??#? ???. -> DROITE, ?#?? .??? -> GAUCHE, ???? ??.? -> HAUT, ??.? ???? -> DROITE, ??? #??? -> GAUCHE, ???? ?.?? -> BAS, ???? #?.? -> GAUCHE, ???? ???? -> GAUCHE, .?. ??.? -> BAS], 12.0

Voici le résultat d'une execution de la classe ArithPopulationMain d'une population de 10 indivdus :

evoluer la population pop
generation : 0
[valeurpropre =X0, fitness = 0.0, valeurpropre =X0, fitness = 0.0, valeurpropre =X1, fitness = 0.0, valeurpropre =X1, fitness = 0.0, valeurpropre =X0, fitness = 0.0, valeurpropre =X0, fitness = 0.0, valeurpropre =X0, fitness = 0.0, valeurpropre =X1, fitness = 0.0, valeurpropre =X0, fitness = 0.0, valeurpropre =X1, fitness = 0.0]
generation : 1
[valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]
generation : 2
[valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]
generation : 3
[valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]
generation : 4
[valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]
generation : 5
[valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]
generation : 6
[valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]
generation : 7
[valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]
generation : 8
[valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]
generation : 9
[valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]

resultat pop finale est : [valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X1, fitness = 212.04658342197766, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737, valeurpropre =X0, fitness = 5.345231574529737]

****

TME7

on souhaite visualiser le déplacement d'un agent dans le labyrinthe on utilisant le dp observer. Dans notre cas c'est la classe Simulation (le sujet) qu'on le souhaite rendre observable, dans cette partie j'ai crée la classe Simulationobs pour ne pas modifier la classe simulation, et l'observateur est notre interface graphique.

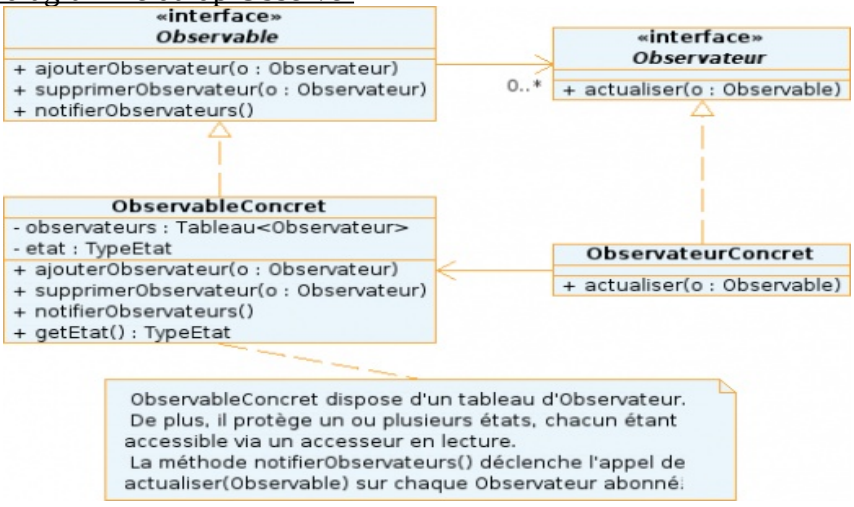
DP Observer : le dp Observer définit une relation entre objets de type un à plusieurs, de sorte que chaque modification du sujet (observable,simulationobs) soit notifiée aux observateurs (interface graphique, LabyActivePanel) afin qu'ils puissent mettre à jour leur état.

Le sujet concret (ObservableConcret) notifie ses observateurs lorsque son état interne est modifié. Lorsqu'un observateur reçoit cette notification, il se met à jour en conséquence.

Le pattern est utilisé dans les cas suivants :

- une modification est utilisé dans l'état d'un objet engendre des modifications dans d'autres objet qui sont déterminés dynamiquement.
- Un objet veut prévenir d'autres objets sans devoir connaître leur type, c'est à dire sans être couplé.
- Lorsqu'on ne veut pas fusionner deux objets en un seul.

voici un diagramme du dp Observer



dans le pacquage pobj.obs on trouve :

-l'interface IsimpleObservable:(Observable) le sujet qui contient les 3 méthodes suivantes : ajouter, rettirer, et mettre à jour les observateurs.

```
public void addOvserver(ISimpleObserver o),public void deleteObserver(ISimpleObserver o),public void notifyObserver();
```

-la classe concrète SimpleObservable(ObsevableConcret) : implemente l'interface IsimpleObservable , introduit l'association avec les observateurs.

-l'interface IsimpleObserver (ObservateurConcret) : est l'interface à implanter pour recevoir des notifications,

```
public void update().
```

On souhaite pouvoir visualiser le comportement du meilleur agent obtenu à l'issue du processus de sélection génétique. Pour cela:

dans le pacquage agent.laby.interf:

- la classe Simulationobs (le sujet): hérite de SimpleObsevable, et à chaque dois que l'agent fait un pas, on informe les observateurs en invoquant la méthode notifyObservers() dont hérite Simulationobs.

Voici le code :

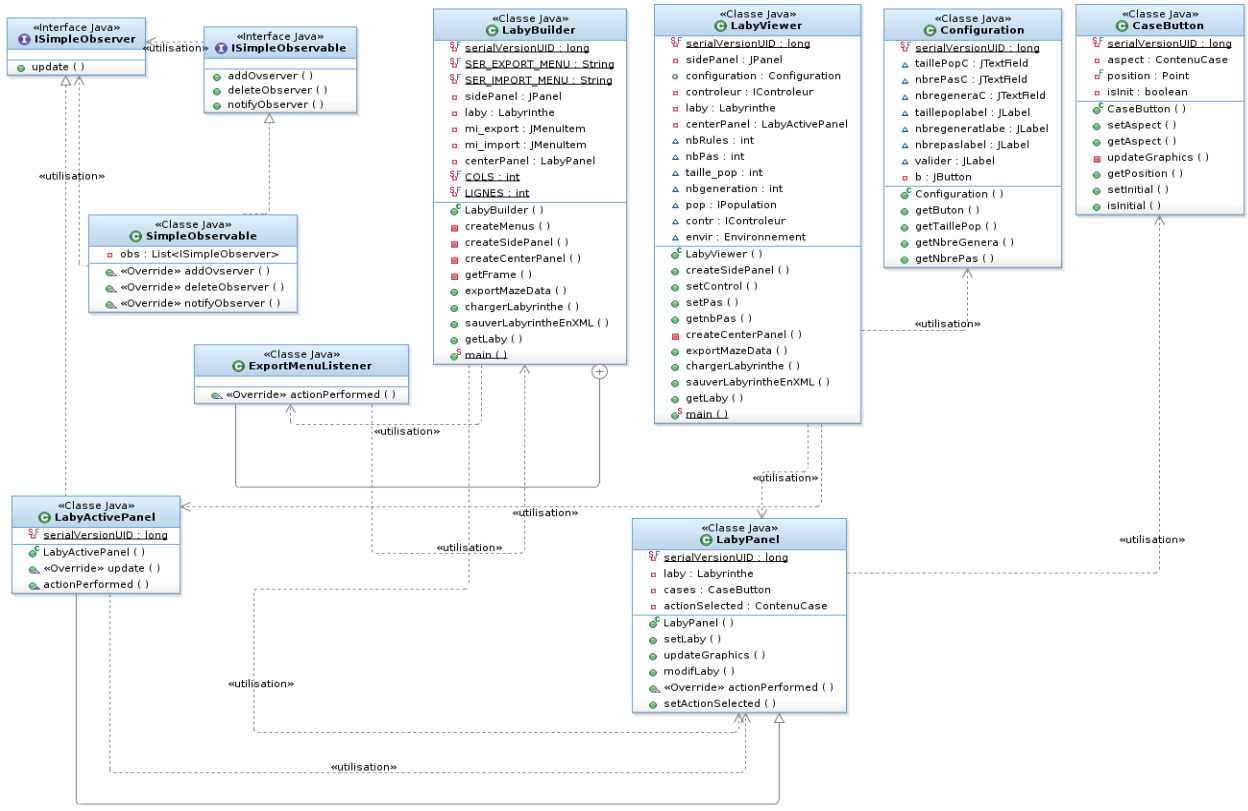
```
public int mesurePerf (int nbPas) {
for (int i=0;i<nbPas;i++){
   agent.faitUnPas(getLaby());
  // mise a jour de observer
    this.notifyObserver();
   }
     return agent.getScore();
}
```

- la classe LabyActivePanel (ObservateurConcret) : implante l'interface IsimpleObserver et dérive du JPanel, elle remplace la classe LabyPanel. met à jour son état avec sa méthode update()(elle force l'affichage à ce rafraîchir en invoquant la méthode updateGraphics().) à chaque fois le sujet lui fournit des données le concernant.

-la classe LabyViewer : permet de visualiser le labyrinthe. Elle contient la methode main .

Après suppression des menus inutiles, j'ai crée une classe Configuration qui hérite de JPanel, cette classe me permet de saisir les données nécessaires, elle contient 4 champs texte et 3 champs de saisies qui sont taille de la population, nombre de pas, nombre de génération et un bouton pour valider les données saisies.

Voici le diagramme du tm7 :



résultat de la simulation du tme7 :

```
Le labyrinthe contient 103 points.
# # # # # # # # # # # # # # # # #
# A . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . #
# # # # # # # # # # # # # # # # #

Le labyrinthe contient 84 points.
# # # # # # # # # # # # # # # # #
#                               #
# . . . . . . . . . . . .       #
# . . . . . . . . . . . .       #
# . . . . . . . . . . . .       #
# . . . . . . . . . . . .       #
# . . . . . . . . . . . .       #
# . . . . . . . . . . . .       #
# . . . . . . . . . . . . A     #
# # # # # # # # # # # # # # # # #
```

****

TME8

on souhaite réaliser notre propre ArrayList contenant des tableaux de taille constante. On l'a nome MyArrayList. L'implémentation de la classe MyArrayList s'appuiera par délégation sur une LinkedList<Vector<T>>.

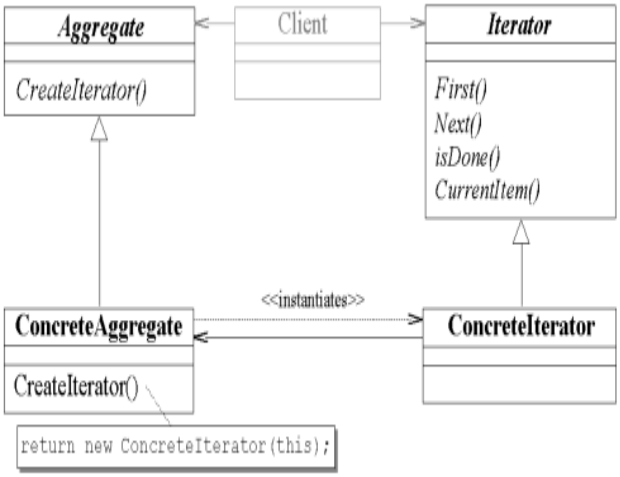
dans notre classe MyArrayList, on implémente les différentes méthodes connues dans Collection<T> suivantes : `public boolean add(T object),public T get(int location),public T set(int location,T object),public int size(),public void setList(LinkedList<Vector<T>> list), public LinkedList<Vector<T>> getList(), public String toString(),public Iterator<T> iterator().`

A qui on ajoute deux constructeurs dont un avec paramètre qui fixe la taille des vecteurs.

`public class MyArrayList<T extends Comparable<T>> extends java.util.AbstractList<T> implements Iterable<T>`

la classe MyArrayList implémente l'interface Iterable<T> dans le but est de pouvoir itérer sur notre LinkedList a l'aide de la foreach, car toute classe qui implémente cette interface peut bénéficier de ce privilège.

DP Iterator : fournit un accès séquentiel à une collection d'objets à des clients sans que ceux ci doivent se préoccuper de l'implantation de cette collection.



on crée notre Iterator<T> (MyIterator) pour la classe MyArrayList. cet Itérateur s'appuiera sur deux itérateurs un sur la liste et l'autre sur les éléments du chaque vecteur.

`private Iterator<Vector<T>> listIT` et `private Iterator<T> vectIT.`

La classe MyIterator <T> implemente l'interface Iterator<T> et ses méthodes `public boolean hasNext(), public T next(), public void remove().`

On rajoute a notre classe MyArrayList un constructeur qui prend une collection en paramètre `public MyArrayList(Collection<T> c).`

on remplace la liste de la population par une MyArrayList, sans oublié de rendre le type T comme sous type de Comparable<T>.

une fois la classe Chrono est branchée, on souhaite comparer la performance de calcule entre les deux versions de la classe Population (celle réalisée au tme2 et tme8) .

Voici les résultats du tme8 pour une population de 500 individus :

```
  le temps necessaire pour la creation (declartion + allocation) d'une population avec une Linkedlist est :
  temps écoulé : 20 millisecondes
le temps necessaire pour l'affichage d'une population avec une Linkedlist est: temps écoulé : 200 millisecondes
la valeur cible de l environnement est : 0.4138841633466649
-evaluation  + tri decroissant + evolution (10 generations) d'une population crée avec une MyrrayList LinkedList
(liste chainée)--------------
le temps necessaire pour evaluer, tri decroissant et evoluer une population avec une Linkedlist est :  temps
écoulé : 13 s 590 millisecondes
le temps final d'une population avec une Linkedlist est :  temps écoulé : 14 s 51 millisecondes
```

Voici les résultats du tme2 pour une population de 500 individus :

```
le temps necessaire pour la creation (declartion + allocation) d'une population avec une ArrayList est :
temps écoulé : 48 millisecondes
le temps necessaire pour l'affichage d'une population avec une ArrayList est :  temps écoulé : 2 millisecondes
le temps necessaire pour evaluer, tri decroissant et evoluer une population avec une ArrayList est :  temps
écoulé : 3 millisecondes
le temps final d'une population avec une ArrayList est :  temps écoulé : 85 millisecondes
```

commentaire :

Une liste chaînée gère une collection de façon ordonnée : l'ajout d'un élément peut se faire au début ou à la fin de la collection. L'ajout d'un élément après n'importe quel élément est lié à la position courante lors d'un parcours :

l'interface qui permet le parcours de la collection est l'interface Iterator (dans le cas d'un iterateur undimentionnel),

et l'interface ListIterator : une sous classe de l'interface Iterator(dans le cas d'un iterateur bidimentionnel)

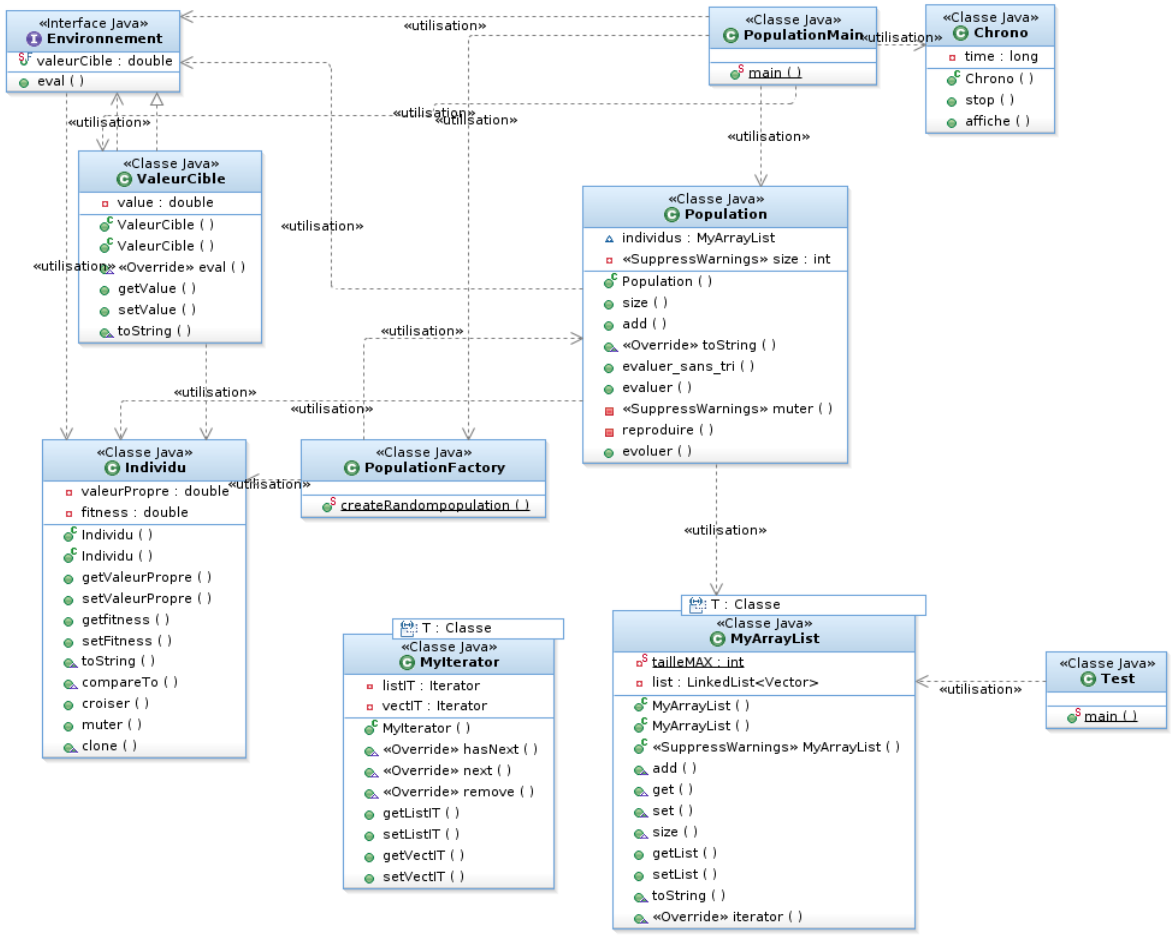
question temps (accés , parcours de la collection et ajout des individus a la collection) :

il n'existe pas de moyen d'obtenir un élément de la liste(linkedlist) directement. Pourtant, la méthode contains() permet de savoir si un élément est contenu dans la liste et la méthode get() permet d'obtenir l'élément à la position fournie en paramètre. Il ne faut toutefois pas oublier que ces méthodes parcourent la liste jusqu'à obtention du résultat, ce qui peut être particulièrement gourmand en terme de temps de réponse.

Une ArrayList est gérée en interne par un tableau. On peut donc accéder en temps constant à n'importe quel élément par get(nb).

Avec une LinkedList, une telle commande est catastrophique en termes de performances (il faut passer par les nb-1 premiers éléments pour accéder au nb-ième

voici diagramme uml du tme8 :



tme9

dans cette partie du tme on souhaite rejouer exactement la même exécution que précédemment, c 'est a dire dans le Tme7 et pour les mêmes valeurs saisies(taille pop=100, pas=30 et nbre Gen= 20) a titre d'exemple et après avoir valider avec le bouton ok. Si on relance le jeu avec le bouton play deux fois de suite, on obtient deux résultats différents : exemple d 'exécution

1ere fois :

```
 # # # # # # # # # # # # # # # # # # # #
# A . . . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . #
# . . . . . . . . . . . . . . . . . . . #
# # # # # # # # # # # # # # # # # # # # #

Le labyrinthe contient 103 points.
# # # # # # # # # # # # # # # # # # # #
#                                       #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . .   #
# . A                                   #
# # # # # # # # # # # # # # # # # # # # #

Le labyrinthe contient 73 points.
```

La 2eme fois : avec les memes valeurs :

```
Le labyrinthe contient 103 points.
# # # # # # # # # # # # # # # # # # # #
#                                       #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . .   #
# . . . . . . . . . . . . . . . . . . A #
# . . . . . . . . . . . . . . . . . . . #
# # # # # # # # # # # # # # # # # # # # #
```

on voit bien que les deux résultats sont différents. C'est dû au séquence des nombres aléatoires qui ont été générer par le programme. Donc il faut les maîtriser.

Pour résoudre ce problème on fait appel au dp singleton, on réalisant notre propre classe de génération de nombres aléatoires qui ne pourra être instanciée qu'une seule fois.

DP Singleton :

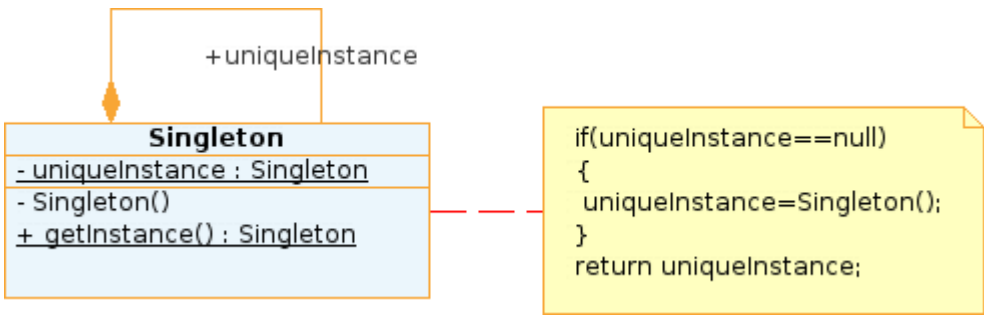
le pattern Singleton a pour but d'assurer qu'une classe ne possède qu'une seule instance et de fournir une méthode de classe unique retournant cette instance.

Dans certains cas, il est utile de gérer des classes ne possédant qu'une seule instance. Dans le cadre des patterns de construction, on trouve aussi la fabrique de produits()dont il n'est pas nécessaire de créer plus d'une instance.

Le pattern est utilisé dans le cas suivant : - il ne doit y avoir qu'une seule instance d'une classe.

- cette instance ne doit être accessible qu'au travers d'une méthode de classe.

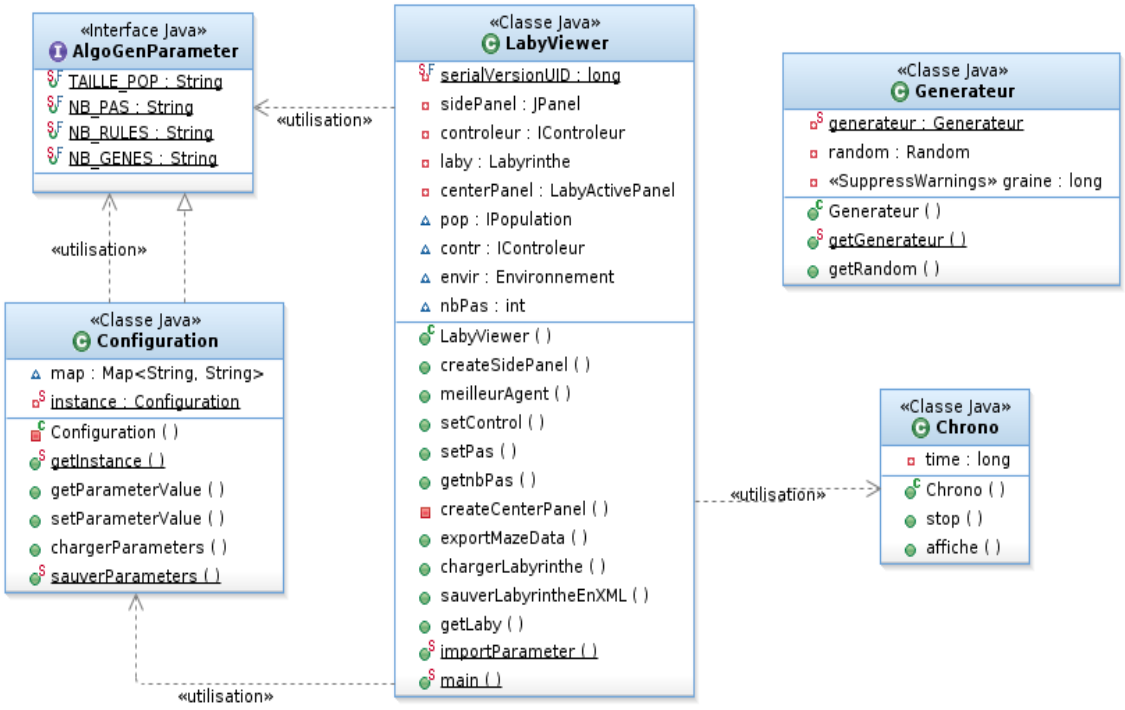
Diagramme UML du design pattern Singleton



se mécanisme est représenté dans la classe Génarteur.

Vu que le code la méthode main conséquent, on va créer un mécanisme qui permet de lire depuis un fichier les données nécessaires.
Dans mon main j'ai nommé ce fichier (« foufa.txt »), il contient la taille de la population, le nombre de pas, nombre de règles et nombre de générations.
On crée une classe Configuration (de la même implantation d'un dp singleton) qui permet de stocker les données dans une map<String, String>. Et de passer ses données a la méthodes main.
La classe Configuration implémente l'interface AlgoGenParameter

voici le diagramme du Tme 9



voici un exmple d'exécution du TME9

```
0.731057369148862
0.731057369148862
# # # # # # # # # # # # # # #
# A . . . . . . . . . . . . #
# . . . . . . . . . . . . . #
# . . . . . . . . . . . . . #
# . . . . . . . . . . . . . #
# . . . . . . . . . . . . . #
# . . . . . . . . . . . . . #
# . . . . . . . . . . . . . #
# . . . . . . . . . . . . . #
# # # # # # # # # # # # # # #

Le labyrinthe contient 103 points.
le temps nécessaire pour une simulation est :
temps écoulé : 11 s 949 millisecondes
# # # # # # # # # # # # # # #
#                           #
#                           #
#                           #
#                           #
#                         A #
#                         . #
#                         . #
#                         . #
# # # # # # # # # # # # # # #

Le labyrinthe contient 3 points.
```

on aura toujours le même jeu d 'exécution.

**********Fin**********

Partie Code

code Tme1

```
 package pobj.algogen;
import java.util.*;
public class Individu {
/**valeur propre pour chaque individu*/
		private double valeurPropre;

		private double fitness = 0;
/** valeur propre aleatoire de individu*/
		public Individu(){
			Random r = new Random();
			valeurPropre = r.nextInt(100);
		}
/** constructeur avec parametre*/
		public Individu(double in){
			valeurPropre=in;
		}
/** accesseur*/
		public double getValeurPropre(){return valeurPropre;}
/** modificateur*/
		public void setValeurPropre(double valeur){
		 valeurPropre= valeur;
		}
/**accesseur*/
		public double getfitness(){return fitness;}
/**modificateur*/
		public void setFitness(double fit){
		 fitness = fit;
		}
/** methode toString de individu*/
		public String toString(){
		 return "[ "+ valeurPropre +", " + fitness + "]";
		}
}
*****************************************************************************************
/** liste d individu*/
  ArrayList<Individu> individus;
 /** la taille de la population*/
  @SuppressWarnings("unused")
private int size = 0;
/** instancier une population*/
  public Population() {
        individus = new ArrayList<Individu>();
  }

  public int size () {
        return individus.size();
  }
 /**ajouter un individu*/
  public void add (Individu individu) {
      individus.add(individu);
  }
  @Override
/** renvoie de la population*/
  public String toString() {
        return Arrays.toString(individus.toArray());
  }
}
*****************************************************************************************************************************************
package pobj.algogen;

public class PopulationFactory{

public static Population createRandompopulation(int size){
                Population pop = new Population() ;
                        for( int i=0; i<size; i++){
                           pop.add (new Individu());
                           }
  return pop;
}
}
*****************************************************************************************
package pobj.algogen;

public class PopulationMain{

		public static void main (String[] args){
                                int i = 10;
                        System.out.println("le nombre d'individu a creer est : " + i);
                System.out.println("la population aleatoire est : " + PopulationFactory.createRandompopulation(i).toString());
        }
}
```

****

code Tme2

```
package pobj.algogen;
import java.util.*;

public class Individu implements Comparable<Individu>{
/**valeur propre pour chaque individu*/
		private double valeurPropre;

		private double fitness = 0;
/** valeur propre aleatoire de individu*/
		public Individu(){
			Random r = new Random();
			valeurPropre = r.nextInt(100);
		}
/** constructeur avec parametre*/
		public Individu(double in){
			valeurPropre=in;
		}
/** accesseur*/
		public double getValeurPropre(){return valeurPropre;}

/** modificateur*/
		public void setValeurPropre(double valeur){
		valeurPropre= valeur;
		}
/**accesseur*/
		public double getfitness(){return fitness;}
/**modificateur*/
		public void setFitness(double fit){
		 fitness = fit;
		}
/** methode toString de individu*/
		public String toString(){
		 return "[ "+ valeurPropre +", " + fitness + "]";
		}
/**methode compareTo*/
		public int compareTo(Individu o){
			if(this.getfitness()>o.getfitness()){
			return 1;
			}else if(this.getfitness()<o.getfitness()){
				return -1;
				}else{
					return 0;
				}
		}
/**croiser l'objet courant, l individu, avec l objet (individu) passé en parametre */
		public Individu croiser(Individu autre){
			Individu nouvelIndividu = new Individu((this.getValeurPropre()+autre.getValeurPropre())/2);
			nouvelIndividu.fitness=this.getfitness();
			return nouvelIndividu;
		}
/** muter l'objet courant avec 5%, changer sa valeur propre, et garder la meme fitness*/
		public void muter(){
			this.valeurPropre = getValeurPropre() * 1.05;
			this.fitness = getfitness();
		}
/**cloner un individu, j ai choisi juste de doublé ses valeurs.*/
		public Individu clone(){

			return new Individu(valeurPropre) ;
			}
}
*******************************************************************************
package pobj.algogen;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class Population {

  /** liste d individu*/
  ArrayList<Individu> individus;

  /** la taille de la population*/
  @SuppressWarnings("unused")
private int size = 0;

  /** instancier une population*/
 public Population() {
       individus = new ArrayList<Individu>();
  }

  public int size () {
       return individus.size();
  }
 /**ajouter un individu*/
  public void add (Individu individu) {
      individus.add(individu);
  }
  @Override
/** renvoie de la population*/
  public String toString() {
```

```
        return Arrays.toString(individus.toArray());
  }
/** methode supplementaire, juste pour mieux voir l'evaluation sans tri*/
  public void evaluer_sans_tri(Environnement cible){
                for(int i =0;i<individus.size();i++){
                        individus.get(i).setFitness(cible.eval(individus.get(i)));
                }
  }
  /** evaluer une population, */
  public void evaluer(Environnement cible){
                for(int i =0;i<individus.size();i++){
                        individus.get(i).setFitness(cible.eval(individus.get(i)));
                }
                for(int i =0;i<individus.size();i++){
                        for(int j =i+1;j<individus.size();j++){
                                if(individus.get(i).compareTo(individus.get(j))<0){
                                        Collections.swap(individus, i, j);
                                }
                        }
                }
  }
  /**muter la population, avec une valeur en parametre*/
  @SuppressWarnings("unused")
private void muter(double m){
                for(int i=0; i< individus.size(); i++)
                individus.get(i).setFitness(individus.get(i).getfitness()+individus.get(i).getfitness()*m);
  }
  /**reproduire une population*/
  private Population reproduire(){
                 /** evaluer la population dans l'environnement cible*/
                Population nouvpop =new Population();
                int taille20prcent = individus.size()/5;
                /** recuperation et clonage des 20% individus  */
                for(int i =0; i<taille20prcent;i++){
                        Individu i2 =individus.get(i).clone();
                        nouvpop.add(i2);
                }
                /** reproduction, croiser et muter les 80% individus*/
                for(int j=0; j<individus.size()-taille20prcent; j++){
        Individu i1, i2, i3;
                        i1 = individus.get(new Random().nextInt(taille20prcent));
                        i2 =individus.get(new Random().nextInt(taille20prcent));
                        i3=i1.croiser(i2);
                        i3.muter();
                        nouvpop.add(i3);
                }
                /** renvoie de la nouvelle generation*/
                return nouvpop;
  }
 /**evolution d'une population dans un environnement*/
  public Population evoluer(Environnement cible){
                  this.evaluer(cible);
                  Population pp = new Population();
                  pp = reproduire();
                  return pp;
  }
}
*********************************************************************************
package pobj.algogen;


public class PopulationFactory{

public static Population createRandompopulation(int size){
                    Population pop = new Population() ;
                                for( int i=0; i<size; i++){
                                    pop.add (new Individu());
                                   }
  return pop;
}
}
*********************************************************************************
package pobj.algogen;

public interface Environnement {

                public double eval(Individu i);
}
*********************************************************************************
package pobj.algogen;

public class ValeurCible implements Environnement {
                private double value;
/**constructeur sans parametre*/
                public ValeurCible(){
                                value = Math.random();
                }
                /** constructeur avec parametre*/
                public ValeurCible(double val){
                                value = val;
                }
/**evaluer un individu dans un environnement, renvoie la fitness*/
                @Override
                public double eval(Individu i) {
                                i.setFitness(1/(Math.pow(getValue() -i.getValeurPropre(),2)));
                                return i.getfitness();
                }
```

```
	/**accesseur*/
	public double getValue(){return value;}
	/**modificateur*/
	public void setValue(double vale){value = vale;}

	public String toString(){
		return "la valeur cible est " + getValue();
	}

}
**********************************************************************************
package pobj.algogen;

public class PopulationMain{

	public static void main (String[] args){
		System.out.println("------------------------TME2-----------------------------------------");

		Chrono time5_TME2 = new Chrono();	// pour le temps total
		Chrono time1_TME2 = new Chrono();				//  le temps de la creation (declaration + allocation) d'une ArrayList
		Population popTME2 = new Population();
		popTME2 =  PopulationFactory.createRandompopulation(10);
		System.out.print("le temps necessaire pour la creation (declartion + allocation) d'une population avec une ArrayList est :  ");
		time1_TME2.stop();

		Chrono time2_TME2 = new Chrono();	// le temps necessaire pour l'afficahge d'une population avec une ArrayList
		System.out.println("la population aleatoire est : " + popTME2);
		System.out.print("le temps necessaire pour l'affichage d'une population avec une ArrayList est :  ");
		time2_TME2.stop();

		Environnement env = new ValeurCible();

		System.out.println("");
		System.out.println("---------evaluation  + tri decroissant + evolution (10 generations) d'une population crée avec une ArrayList----------");

		Chrono time3_TME2 = new Chrono(); // le temps necessaire pour evaluer tri decroissant et evoluer une population crée avec une ArrayList
		popTME2.evaluer(env);
		for (int i =1; i <=10; i++){
		popTME2 = popTME2.evoluer(env);
		}
		System.out.print("le temps necessaire pour evaluer, tri decroissant et evoluer une population avec une ArrayList est :  ");
		time3_TME2.stop();
		System.out.println(popTME2);
		System.out.print("le temps final d'une population avec une ArrayList est :  ");
		time5_TME2.stop();

		}
	}
**********************************************************************************************************************************************************
```

---

code Tme3

```
package pobj.arith;

public interface Expression {
	public double eval(EnvEval env);

}
**********************************************************************************************
package pobj.arith;

public class Constante implements Expression {
	/**
	 * valeur de la constante
	 */
	private double value;
	/**constructeur avec une valeur en parametre*/
	public Constante(double val){
		value = Double.valueOf(val);
	}
	/**renvoie la valeur de la constante*/
	public double getValue(){
		return value;
	}
	/**la methode toString de l'objet courant*/
	public String tostring(){
		String a= String.valueOf(getValue());
		return a;
	}
	@Override
	/**la methode d'evaluation de l'objet dans un envieronnement,
	 *  elle renvoie toujours la valeur de la constante qul que soit l'environnement*/
	public double eval(EnvEval env) {
		return getValue();
	}
}
**********************************************************************************************
package pobj.arith;

public class Variable implements Expression {
   /**
```

```
	*  rang de la variable dans la liste
	*/
		private int rang;
		/**
		 * constructeur
		 */
		public Variable (int a){
			rang = a;
		}
		@Override
		public double eval(EnvEval env) {
			return env.getValue(rang) ;
		}
		public String toString(){
			return "X" + rang;
		}
}**********************************************************************************************
package pobj.arith;

import java.util.Arrays;

public class EnvEval {

		private double variables[];

		public EnvEval(int taille){
			variables = new double[taille];
		}

		public void setVariable(int indexVariable, double nouvellevaleur){
			variables[indexVariable] =nouvellevaleur;
		}
		@Override
		public String toString(){
			return Arrays.toString(variables);
		}

		public double getValue(int indexVariable){
			return variables[indexVariable];
		}
}
**********************************************************************************************
package pobj.arith;

public class OperateurBinaire implements Expression {
		/**
		 *les variables
		 *@param type, type de l'operande
		 *@param left, right, operande gauche et droite
		 **/
		Operator type;
		private Expression left,right;
		/**
		 * constructeur
		 * */
		public OperateurBinaire (Operator op, Expression exp0, Expression exp1){
					type = op;
					left= exp0;
					right = exp1;
		}
		/**renvoie l'expression gauche*/
public Expression getLeft(){
			return left;
			}
		/**renvoie l'expression droite*/
public Expression getRight(){
			return right;
		}
		/**renvoie le type*/
public Operator getType(){
			return type;
		}
		/**l'evaluation de l'expression */
		@Override
public double eval(EnvEval env) {

			switch (type){
			case PLUS :
								return getLeft().eval(env) + getRight().eval(env);
			case DIV :
								return getLeft().eval(env) / getRight().eval(env);
			case MINUS :
						return getLeft().eval(env) - getRight().eval(env);
			case MULT :
					return getLeft().eval(env) * getRight().eval(env);
```

```
                default :
                                        System.out.println("blablablabla");

                }
                return 0;
        }
        /**
         * affichage d'une expression
         */
        public String toString(){
                StringBuilder chaine = new StringBuilder( "("+ left);
                        switch (type){
                        case PLUS :
                                chaine.append(" + ");
                                break;
                        case MINUS :
                                chaine.append(" - ");
                                break;
                        case MULT :
                                chaine.append(" * ");
                                break;
                        default :
                                chaine.append(" / ");
                                break;
                        }

                        chaine.append(right + ")");

                return chaine.toString();
        }
}
**********************************************************************************************
package pobj.arith;

import java.util.Random;

public class Expressionfactory {

        private static int MAXVARIABLE = 2;
        private static Random generateur= new Random();
        /**
         * Un constructeur pour des expressions binaires usuelles: +,-,*,/
         * @param op le type de l'opérande, {@link Operator}, PLUS,MOINS,MULT,DIV
         * @param left operande gauche
         * @param right operande droite
         * @return une expression binaire
         */

        public static Expression createOperateurbinaire(Operator op, Expression ex, Expression exp){
                        return new OperateurBinaire(op,ex,exp);
        }
        /**
         * Un constructeur d'expressions constantes.
         * @param constant sa valeur
         * @return une constante
         */
        public static Expression createConstant(double constant){
                return new Constante(constant);
        }
        /**
         * Un constructeur de variables, identifiées par un entier compris entre 0 et MAXVARIABLES.
         * La demande de création de variables d'indice plus grand entraine un accroissement de
         * MAXVARIABLE (attribut static).
         * @param id l'indice de la variable
         * @return une Variable
         */
        public static Expression createVariable(int id){
                if(id > MAXVARIABLE)
                        MAXVARIABLE+= 1;
                return new Variable(id);
        }


        public static Expression createRandomExpression(){
                return createRandomExpression(3);
        }

        /**
         * creation d'une expression aleatoire avec limitation de la profondeur
         * @param profondeur
         * */
        public static Expression createRandomExpression(int profondeur){
                int type =0;
                if(profondeur == 0){
                        type= generateur.nextInt(2);
```

```
		}else{
			type= generateur.nextInt(3);
			}
		switch(type){
		case 0:
			return createVariable(generateur.nextInt(MAXVARIABLE));
		case 1 :
			return createConstant(generateur.nextDouble());
		default :
			Expression left = createRandomExpression(profondeur-1);
			Expression right = createRandomExpression(profondeur-1);

			return createOperateurbinaire(Operator.values()[type],left,right);
		}
	}

	/**
	 * Génère un environnement d'évaluation aléatoire, en supposant qu'il n'y
	 * a pas plus de MAXVARIABLES.
	 * @return Un environnement généré aléatoirement.
	 */

	public static EnvEval createRandomEnvironment(){
		EnvEval env = new EnvEval( MAXVARIABLE);
		for(int i=0;i< MAXVARIABLE; i++)
			env.setVariable(i, generateur.nextDouble());
		return env;

	}
}
*****************************************************************************************
package pobj.arith;

public class Testexpression {
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		System.out.println("----------teste de la classe EnvEval---------");
		EnvEval e = new EnvEval(10);
		System.out.println(e.toString());
		e.setVariable(3, 4);
		System.out.println(e.toString());
		System.out.println("");

		System.out.println("----------teste de la classe constante ----------");
		Constante cst = new Constante(10);
		double a =cst.eval(e);
		System.out.println("la valeur de la constante est :"+a);
		System.out.println(cst.tostring());
		System.out.println("");

		System.out.println("----------teste de la classe Variable ----------");
		Variable var = new Variable(3);
		double dou =var.eval(e);
		System.out.println("la valeur de la variable "+var.toString()+" est : "+dou);
		System.out.println("");

		System.out.println ("----------teste de la classe OperateurBinaire----------");
		Operator a1 = Operator.PLUS;
		Expression e1 = var;
		Expression e2 = var;
		Expression expr1 = new OperateurBinaire(a1, e1,e2);
		System.out.println("expression : " + expr1.toString());
		double valeur =expr1.eval(e);
		System.out.println("le resultat de l'expresion est : " + valeur);
		System.out.println("");

		System.out.println ("----------teste de la classe Expressionfactory ------");
		System.out.print("exemple d'un environnement aleatoire : ");
		EnvEval env1 = Expressionfactory.createRandomEnvironment();
		System.out.println(env1);
		System.out.println(" ");
		System.out.println("**********************************************************");
		System.out.println("evoluer une expression binaire simple :");
		EnvEval en1 = new EnvEval(2);
		en1.setVariable(0,5);
		Expression ex1 = new Variable(0);
		en1.setVariable(1,15);
		Expression ex2 = new Variable(1);
		Expression opra1 =Expressionfactory.createOperateurbinaire(Operator.PLUS, ex1, ex2);
		System.out.print(opra1.toString()+ " = "+ opra1.eval(en1));
		System.out.println(" ");
```

```
		System.out.println("*****************************************************");
		System.out.println("evaluer une expression binaire avec profondeur :");

		Expression exprofondeur=Expressionfactory.createRandomExpression(3);
		EnvEval envprofondeur = Expressionfactory.createRandomEnvironment();
		System.out.println(envprofondeur);
		System.out.println(exprofondeur.toString() + " = " + exprofondeur.eval(envprofondeur));


	}
}
```

****

code Tme4

pacquage pobj.algogen :

```
package pobj.algogen;

public interface Environnement {

	public double eval(Individu i);
}
```

****

```
package pobj.algogen;
import pobj.arith.Expression;
import pobj.arith.Expressionfactory;
import pobj.arith.Operator;

public class Individu implements Comparable<Individu>{
/**
 * @param valeur propre de type Expression pour chaque individu*/
	private Expression valeurPropre;

	private double fitness = 0;
/** valeur propre aleatoire de individu*/
	public Individu(){
		this(Expressionfactory.createRandomExpression());
	}
/** constructeur avec parametre*/
	public Individu(Expression in){
		valeurPropre=in;
	}
/** accesseur*/
	public Expression getValeurPropre(){return valeurPropre;}

/** modificateur*/
	public void setValeurPropre(Expression valeur){
	valeurPropre= valeur;
	 }
/**accesseur*/
	public double getfitness(){return fitness;}
/**modificateur*/
	public void setFitness(double fit){
	 fitness = fit;
	}
/** methode toString de individu*/
	public String toString(){
	 return "[ "+ valeurPropre +", " + fitness + "]";
	}
/**methode compareTo*/
	public int compareTo(Individu o){
		if(this.getfitness()>o.getfitness()){
		return 1;
		}else if(this.getfitness()<o.getfitness()){
			return -1;
			}else{
				return 0;
			}
	}
/**croiser l'objet courant, l individu, avec l objet (individu) passé en parametre */
	public Individu croiser(Individu autre){
		if(this.valeurPropre.equals(autre.valeurPropre)){
			return new Individu(this.valeurPropre);
		}
		return new Individu(
				Expressionfactory.createOperateurbinaire(Operator.DIV,
					Expressionfactory.createOperateurbinaire(Operator.PLUS,

this.getValeurPropre(),autre.getValeurPropre()),Expressionfactory.createConstant(2)).simplifier());
	}

/** muter l'objet courant avec 5%, changer sa valeur propre, et garder la meme fitness*/
	public void muter(){
		this.valeurPropre = Expressionfactory.createRandomExpression();
		this.fitness = getfitness();
	}
/**cloner un individu.*/
	public Individu clone(){
		return new Individu(getValeurPropre());
		}
}
```

****

```
package pobj.algogen;
import java.util.Arrays;
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class Population {

  /** liste d individu*/
  ArrayList<Individu> individus;

  /** la taille de la population*/
  @SuppressWarnings("unused")
private int size =0;

  /** instancier une population*/
 public Population() {
        individus = new ArrayList<Individu>();
  }

  public int size () {
        //return size;
        return individus.size();
  }
 /**ajouter un individu*/
  public void add (Individu individu) {
       individus.add(individu);
  }
  @Override
/** renvoie de la population*/
  public String toString() {
        return Arrays.toString(individus.toArray());
  }
/** methode supplementaire, juste pour mieux voir l'evaluation sans tri*/
  public void evaluer_sans_tri(Environnement cible){
                for(int i =0;i<individus.size();i++){
                        individus.get(i).setFitness(cible.eval(individus.get(i)));
                }
  }
  /** evaluer une population, */
  public void evaluer(Environnement cible){
                for(int i =0;i<individus.size();i++){
                        individus.get(i).setFitness(cible.eval(individus.get(i)));
                }
                for(int i =0;i<individus.size();i++){
                        for(int j =i+1;j<individus.size();j++){
                                if(individus.get(i).compareTo(individus.get(j))<0){
                                        Collections.swap(individus, i, j);
                                }
                        }
                }
  }
  /**muter la population, avec une valeur en parametre*/
  private void muter(double m){
                for(int i=0; i< individus.size(); i++)
               individus.get(i).setFitness(individus.get(i).getfitness()+individus.get(i).getfitness()*m);
  }
  /**reproduire une population*/
  private Population reproduire(){
                 /** evaluer la population dans l'environnement cible*/
                Population nouvpop =new Population();
                int taille20prcent = individus.size()/5;
                /** recuperation et clonage des 20% individus  */
                for(int i =0; i<taille20prcent;i++){
                        Individu i2 =individus.get(i).clone();
                        nouvpop.add(i2);
                }
                /** reproduction, croiser et muter les 80% individus*/
                for(int j=0; j<individus.size()-taille20prcent; j++){
        Individu i1, i2, i3;
                        i1 = individus.get(new Random().nextInt(taille20prcent));
                        i2 =individus.get(new Random().nextInt(taille20prcent));
                        i3=i1.croiser(i2);
                        i3.muter();
                        nouvpop.add(i3);
                }
                /** renvoie de la nouvelle generation*/
                return nouvpop;
  }
 /**evolution d'une population dans un environnement*/
  public Population evoluer(Environnement cible){
                 this.evaluer(cible);
                 Population pp = new Population();
                 pp = reproduire();
                 pp.evaluer(cible);
                 return pp;
  }
}
*********************************************************************************
package pobj.algogen;

import pobj.arith.EnvEval;
import pobj.arith.Expressionfactory;

public class ValeurCible implements Environnement {
                private double value;
                private EnvEval env;
/**constructeur sans parametre*/
                public ValeurCible(){
```

```
                        value = Math.random();
                        env = Expressionfactory.createRandomEnvironment();
                }
/**evaluer un individu dans un environnement,
 * renvoie la fitness*/
                @Override
                public double eval(Individu i) {
                        double r1= i.getValeurPropre().eval(env);
                        return 1/((value -r1)*(value-r1));
                }
                /**accesseur*/
                public double getValue(){return value;}
                /**modificateur*/
                //public void setValue(double vale){value = vale;}

                public String toString(){
                        return "je cherche la valeur : " + getValue();
                }
}
*******************************************************************************
package pobj.algogen;

public class PopulationFactory{

public static Population createRandompopulation(int size){
                Population pop = new Population() ;
                        for( int i=0; i<size; i++){
                            pop.add (new Individu());
                           }
  return pop;
}
}
*******************************************************************************
package pobj.algogen;

public class PopulationMain{

        public static void main (String[] args){
        Population popu =  PopulationFactory.createRandompopulation(20);

                Environnement a = new ValeurCible();
                popu.evaluer_sans_tri(a);
                System.out.println("-----------------------------population sans tri--------------------------------------");
                System.out.println("la population des individus [valeurPropre, fitness] apres evaluation,sans tri decroissant
par fitness ,  : " );
                System.out.println(popu.toString());
                System.out.println("");
                popu.evaluer(a);
                System.out.println("-----------------------------population avec tri
decroissant---------------------------------------");
                System.out.println("la population des individus [valeurPropre, fitness] apres evaluation,tri decroissant par
fitness ,  : " );
                System.out.println(popu.toString());
                System.out.println("");
                for (int i =1; i <=10; i++){
                System.out.println("-----------------------------population evoluer, generation  : " +i +
"-----------------------------------");
                popu = popu.evoluer(a);
                System.out.println(popu);
                popu = popu.evoluer(a);
                }
        }
}
*******************************************************************************
```

package pobj.arith :

```
 package pobj.arith;

public class Constante implements Expression {
        /**
         * valeur de la constante
         */
        private double value;
        /**constructeur avec une valeur en parametre*/
        public Constante(double val){
                value = val;
        }
        /**renvoie la valeur de la constante*/
        public double getValue(){
                return value;
        }
        /**la methode toString de l'objet courant*/
        public String tostring(){

                return ""+value ;
        }
        @Override
        /**la methode d'evaluation de l'objet dans un envieronnement,
         *  elle renvoie toujours la valeur de la constante qul que soit l'environnement*/
        public double eval(EnvEval env) {
                return getValue();
        }
        @Override
        public Expression simplifier() {
                return Expressionfactory.createConstant(value);
        }
```

```
	@Override
	public Expression clone(){
		return new Constante(value);
	}

}
```

```
package pobj.arith;

import java.util.Arrays;

public class EnvEval {

	private double variables[];

	public EnvEval(int taille){
		variables = new double[taille];
	}

	public void setVariable(int indexVariable, double nouvellevaleur){
		variables[indexVariable] =nouvellevaleur;
	}
	@Override
	public String toString(){
		return Arrays.toString(variables);
	}

	public double getValue(int indexVariable){
		return variables[indexVariable];
	}
}
```

```
package pobj.arith;

public interface Expression {
	public double eval(EnvEval env);
	public Expression clone();
	public Expression simplifier();
}
```

```
package pobj.arith;

public class Variable implements Expression {
    /**
     *  rang de la variable dans la liste
     */
	private int rang;
	/**
	 * constructeur
	 */
	public Variable (int a){
		rang = a;
	}

	@Override
	public double eval(EnvEval env) {
		return env.getValue(rang) ;
	}

	public String toString(){
		return "X" + rang;
	}
	@Override
	public Expression simplifier() {
		return Expressionfactory.createVariable(rang);
	}
	public Expression clone(){
		return new Variable(rang);
	}
}
```

```
package pobj.arith;

public class OperateurBinaire implements Expression {
	/**
	 *les variables
	 *@param type, type de l'operande
	 *@param left, right, operande gauche et droite
	 **/
	Operator type;
	private Expression left,right;
	/**
	 * constructeur
	 * */
	public OperateurBinaire (Operator op, Expression exp0, Expression exp1){
			type = op;
			left= exp0;
			right = exp1;
	}
	/**renvoie l'expression gauche*/
public Expression getLeft(){
		return left;
		}
	/**renvoie l'expression droite*/
public Expression getRight(){
		return right;
```

```
        }
        /**renvoie le type*/
public Operator getType(){
                return type;
        }
        /**l'evaluation de l'expression */
        @Override
public double eval(EnvEval env) {

                switch (type){
                case PLUS :
                                        return getLeft().eval(env) + getRight().eval(env);
                case DIV :
                                        return getLeft().eval(env) / getRight().eval(env);
                case MINUS :
                                    return getLeft().eval(env) - getRight().eval(env);
                case MULT :
                                return getLeft().eval(env) * getRight().eval(env);
                default :
                                    System.out.println("blablablabla");

                }
                return 0;
        }
        /**
         * affichage d'une expression
         */
        public String toString(){
                StringBuilder chaine = new StringBuilder( "("+ left);
                        switch (type){
                        case PLUS :
                                chaine.append(" + ");
                                break;
                        case MINUS :
                                chaine.append(" - ");
                                break;
                        case MULT :
                                chaine.append(" * ");
                                break;
                        default :
                                chaine.append(" / ");
                                break;
                        }

                        chaine.append(right + ")");
                return chaine.toString();
        }
        @Override
        public Expression simplifier() {
                Expression lefts = left.simplifier();
                Expression rights = right.simplifier();
                if (rights instanceof Constante && lefts instanceof Constante) {
                        return Expressionfactory.createConstant(eval(null));
                } else {
                        return Expressionfactory.createOperateurbinaire(type, lefts, rights);
                }
        }

        public Expression clone(){
                return new OperateurBinaire(type, left.clone(),right.clone());
        }
}
*****************************************************************************************************************************************************
package pobj.arith;
import java.util.Random;

public class Expressionfactory {

        private static int MAXVARIABLE = 2;
        private static Random generateur= new Random();
        /**
         * Un constructeur pour des expressions binaires usuelles: +,-,*,/
         * @param op le type de l'opérande, {@link Operator}, PLUS,MOINS,MULT,DIV
         * @param left operande gauche
         * @param right operande droite
         * @return une expression binaire
         */

        public static Expression createOperateurbinaire(Operator op, Expression ex, Expression exp){
                        return new OperateurBinaire(op,ex,exp);
        }
        /**
         * Un constructeur d'expressions constantes.
         * @param constant sa valeur
         * @return une constante
         */
        public static Expression createConstant(double constant){
                return new Constante(constant);
        }
        /**
         * Un constructeur de variables, identifiées par un entier compris entre 0 et MAXVARIABLES.
         * La demande de création de variables d'indice plus grand entraine un accroissement de
         * MAXVARIABLE (attribut static).
         * @param id l'indice de la variable
         * @return une Variable
         */
```

```java
	public static Expression createVariable(int id){
		if(id > MAXVARIABLE)
			MAXVARIABLE=id + 1;
		return new Variable(id);
	}


	public static Expression createRandomExpression(){
		return createRandomExpression(3);
	}

	/**
	 * creation d'une expression aleatoire avec limitation de la profondeur
	 * @param profondeur
	 * */
	public static Expression createRandomExpression(int profondeur){
		int type =0;
		int ope = 0;
		if(profondeur == 0){
			type= generateur.nextInt(2);
		}else{
			ope= generateur.nextInt(4);
			}
		switch(type){
		case 0:
			return createVariable(generateur.nextInt(MAXVARIABLE));
		case 1 :
			return createConstant(generateur.nextDouble());
		default :
			Expression left = createRandomExpression(profondeur-1);
			Expression right = createRandomExpression(profondeur-1);

			return createOperateurbinaire(Operator.values()[ope],left,right);
		}
	}

	/**
	 * Génère un environnement d'évaluation aléatoire, en supposant qu'il n'y
	 * a pas plus de MAXVARIABLES.
	 * @return Un environnement généré aléatoirement.
	 */

	public static EnvEval createRandomEnvironment(){
		EnvEval env = new EnvEval( MAXVARIABLE);
		for(int i=0;i< MAXVARIABLE; i++)
			env.setVariable(i, generateur.nextDouble());
		return env;

	}
}
```

```java
package pobj.arith;

public class Testexpression {
	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		System.out.println("--------------teste de la classe EnvEval---------");
		EnvEval e = new EnvEval(10);
		System.out.println(e.toString());
		e.setVariable(3, 4);
		System.out.println(e.toString());
		System.out.println("");

		System.out.println("------------------teste de la classe constante --------------");
		Constante cst = new Constante(10);
		double a =cst.eval(e);
		System.out.println("la valeur de la constante est :"+a);
		System.out.println(cst.tostring());
		System.out.println("");

		System.out.println("------------------teste de la classe Variable ------------");
		Variable var = new Variable(3);
		double dou =var.eval(e);
		System.out.println("la valeur de la variable "+var.toString()+" est : "+dou);
		System.out.println("");

		System.out.println ("---------------teste de la classe OperateurBinaire----------------------------------");
		Operator a1 = Operator.PLUS;
		Expression e1 = var;
		Expression e2 = var;
		Expression expr1 = new OperateurBinaire(a1, e1,e2);
		System.out.println("expression : " + expr1.toString());
		double valeur =expr1.eval(e);
		System.out.println("le resultat de l'expresion est : " + valeur);
		System.out.println("");

		System.out.println ("------------teste de la classe Expressionfactory ------");
		System.out.print("exemple d'un environnement aleatoire : ");
		EnvEval env1 = Expressionfactory.createRandomEnvironment();
		System.out.println(env1);
		System.out.println(" ");
		System.out.println("**************************************************");
		System.out.println("evoluer une expression binaire simple :");
```

```
                EnvEval en1 = new EnvEval(2);
                en1.setVariable(0,5);
                Expression ex1 = new Variable(0);
                en1.setVariable(1,15);
                Expression ex2 = new Variable(1);
                Expression opra1 =Expressionfactory.createOperateurbinaire(Operator.PLUS, ex1, ex2);
                System.out.print(opra1.toString()+ " = "+ opra1.eval(en1));
                System.out.println(" ");

                System.out.println("*************************************************");
                System.out.println("evoluer une expression binaire avec profondeur :");

                Expression exprofondeur=Expressionfactory.createRandomExpression(8);
                EnvEval envprofondeur = Expressionfactory.createRandomEnvironment();
                System.out.println(envprofondeur);
                System.out.println(exprofondeur.toString() + " = " + exprofondeur.eval(envprofondeur));


        }
}
```

****

TME5

package exception :

```
  package exception;

import java.awt.Point;

public class LabyErroneException extends Exception {
        /**
         *
         */
        private static final long serialVersionUID = 1L;
        private Point point;
        public LabyErroneException (Point p, String message){
                super (message);
                point = p;
        }
        public Point getPoint(){
                return point;
        }
}
```

****

```
package exception;

import java.awt.Point;
import agent.laby.ContenuCase;
import agent.laby.Labyrinthe;


public class VerificationLaby {

        public static void verifierconditions(Labyrinthe l)throws LabyErroneException{
                estCaseInitialevide(l);
                estEntoureDeMurs(l);
        }

        /** verifier si la case depart est bien vide, sinon lever une exception*/
        public static void estCaseInitialevide(Labyrinthe l)throws CaseDepartNonVideException{
                Point p = l.getPositionInitiale();
                if (l.getContenuCase(p) != ContenuCase.VIDE)
                        throw new CaseDepartNonVideException(p);
        }

   public static void estEntoureDeMurs(Labyrinthe l)throws LabyMalEntoureException{
        Point p = l.estEntoureDeMurs();
                if (p != null)
                        throw new LabyMalEntoureException(p);
        }

   public static int corrigerLabyrinthe(Labyrinthe l) throws LabyErroneException{
        int nbrerr = 0;
         while(true){

                try{
                        verifierconditions(l);
                         return  nbrerr;
                }catch(CaseDepartNonVideException e){
                        nbrerr++;
                        l.setContenuCase(e.getPoint(), ContenuCase.VIDE);
                        System.out.println(e.getPoint());
                }catch(LabyMalEntoureException e){
                        nbrerr++;
                        l.setContenuCase(e.getPoint(), ContenuCase.MUR);
                        System.out.println(e.getPoint());
                }
        }
 }
}
```

****

```
package exception;

import java.awt.Point;
```

```java
public class CaseDepartNonVideException extends LabyErroneException {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	public CaseDepartNonVideException(Point p) {
		super(p, "la case depart de labyrinthe doit etre vide.merci");
		// TODO Auto-generated constructor stub
	}

}
```

****

```java
package exception;

import java.awt.Point;


public class LabyMalEntoureException extends LabyErroneException {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	public LabyMalEntoureException(Point p) {
		super(p,"labyrinthe mal entouré, le point doit representer un mur");
	}
}
```

****

dans le package agent :

```java
package agent;
import java.io.IOException;
import exception.LabyErroneException;

import exception.VerificationLaby;

import agent.control.ControlFactory;
import agent.control.IControleur;
import agent.laby.ChargeurLabyrinthe;
import agent.laby.Labyrinthe;


public class CorrectionErreur {
	/**
	 * @param args[0] : nom du fichier contenant le labyrinthe
	 * @param args[1] : nombre de pas d'évaluation
	 * @param args[2] : nombre de règles par controleur
	 * @throws LabyErroneException
	 * @throws IOException
	 */
	public static void main(String[] args){

		String labyFile = "foufi.mze"; // args[0];
		String labyFile1 = "foufi1.mze";
		int nbSteps = 50; // Integer.parseInt(args[1]);
		int nbRules = 10; // Integer.parseInt(args[2]);
		try {
			int i;
			System.out.println("exemple erreur pour le fichier foufi");
			Labyrinthe laby = ChargeurLabyrinthe.chargerLabyrinthe(labyFile);
			i=VerificationLaby.corrigerLabyrinthe(laby);
			System.out.println("le nbre d'erreur est : " +i);
			ChargeurLabyrinthe.sauverLabyrinthe("fouficorrectionlaby.mze", laby);
			System.out.println("exemple erreur pour le fichier foufi1");
			Labyrinthe laby1 = ChargeurLabyrinthe.chargerLabyrinthe(labyFile1);
			i=VerificationLaby.corrigerLabyrinthe(laby1);
			System.out.println("le nbre d'erreur est : " +i);
			ChargeurLabyrinthe.sauverLabyrinthe("foufi1correctionlaby.mze", laby);

			IControleur sc = ControlFactory.createControleur(nbRules);
			Simulation sim = new Simulation(laby, sc);
			System.out.println(sim.mesurePerf(nbSteps));
			System.out.println(sc);


		} catch (IOException e) {
			System.out.println("Problème de chargement du labyrinthe"+e);
			System.exit(1);
		}catch (LabyErroneException e) {
			System.out.println("ops, y a des erreurs , tkt, je corrige"+e);
			e.printStackTrace();
		}
	}
}
```

****

```java
package agent;

import java.io.IOException;
import exception.LabyErroneException;

import agent.control.ControlFactory;
import agent.control.IControleur;
import agent.laby.ChargeurLabyrinthe;
import agent.laby.Labyrinthe;
```

```
/**
 * Classe principale pour tester le comportement des agents dans
 *        le labyrinthe.
 *  @author sigaud
 */
public class SimuMain {

        /**
         * @param args[0] : nom du fichier contenant le labyrinthe
         * @param args[1] : nombre de pas d'évaluation
         * @param args[2] : nombre de règles par controleur
         * @throws LabyErroneException
         * @throws IOException
         */
        public static void main(String[] args){

                String labyFile = "goal.mze"; // args[0];
                int nbSteps = 50; // Integer.parseInt(args[1]);
                int nbRules = 10; // Integer.parseInt(args[2]);
                try {

                        Labyrinthe laby = ChargeurLabyrinthe.chargerLabyrinthe(labyFile);
                        IControleur sc = ControlFactory.createControleur(nbRules);
                        //IControleur sc = ControlFactory.createControleurDroitier();

                        Simulation sim = new Simulation(laby, sc);
                        System.out.println(sim.mesurePerf(nbSteps));
                        //System.out.println(sim.mesurePerf(20));    // le nbre de case libre est de 12
                        System.out.println(sc);


                } catch (IOException e) {
                        System.out.println("Problème de chargement du labyrinthe"+e);
                        System.exit(1);
                }
        }
}
```

****

dans le package test.agent :

```
package test.agent;

import java.io.IOException;

import junit.framework.TestCase;
import agent.Simulation;
import agent.control.ControlFactory;
import agent.control.IControleur;
import agent.laby.ChargeurLabyrinthe;
import agent.laby.Labyrinthe;

public class AgentTest extends TestCase {
        Labyrinthe laby= null, laby1=null, laby2=null;
        private Simulation simuTest,simuTest1,simuTest2;
        String labyFile = "goal.mze";
        String labyFile1 = "sim1.mze";
        String labyFile2 = "sim2.mze";
protected void setUp() throws Exception {
                super.setUp();
                try{
                        laby = ChargeurLabyrinthe.chargerLabyrinthe(labyFile);
                        laby1 = ChargeurLabyrinthe.chargerLabyrinthe(labyFile1);
                        laby2 = ChargeurLabyrinthe.chargerLabyrinthe(labyFile2);
                }catch (IOException e) {
                        System.out.println(" problème de chargement du labyrinthe " + e.getMessage());
                }
                IControleur sc = ControlFactory.createControleurDroitier();

                simuTest=new Simulation(laby, sc);
                simuTest1=new Simulation(laby1, sc);
                simuTest2=new Simulation(laby2, sc);

        }

        public void testAgent() {
                System.out.println("le score est : " +simuTest.mesurePerf(20));
                System.out.println("le score est : " +simuTest1.mesurePerf(20));
                System.out.println("le score est : " +simuTest1.mesurePerf(20));
        }
        public void testMesurePerf(){
    int score = simuTest.mesurePerf(20);
    assertTrue(score == 12);        //assertTrue(boolean condition)
                                                                        //Affirme qu'une condition est vraie, prend la couleur verte. rouge sinon
    assertFalse(score == 9);                //assertFalse (boolean condition)
                                                                        //Affirme qu'une condition est fausse. de meme couleur verte, rouge sinon

    int score1 = simuTest1.mesurePerf(20);
    assertTrue(score1 == 2);
    assertFalse(score1 == 9);

    int score2 = simuTest2.mesurePerf(20);
    assertTrue(score2 == 2);
    assertFalse(score2 == 12);

        }
}
```

****

TME6

package pobj.algogen :

```
 package pobj.algogen;

public interface Environnement {

        public double eval(Individu i);
}
*******************************************************************************
package pobj.algogen;

public interface IPopulation extends Iterable<Individu> {

        public abstract IPopulation evoluer(Environnement cible);

        public void add(Individu individu);
}
*******************************************************************************
package pobj.algogen;

public interface Individu extends Comparable<Individu>{
        public void muter();
        public Individu croiser(Individu autre);
        public double getFitness();
        public void setFitness(double fit);
        public Individu clone();
}
*******************************************************************************************
package pobj.algogen;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.Random;

public class Population implements IPopulation{
 /** liste d individu*/
 ArrayList<Individu> individus = new ArrayList<Individu>();

 /** la taille de la population*/
 public int size () {
      return individus.size();
 }
/**ajouter un individu*/
 public void add (Individu individu) {
     individus.add(individu);
 }
 @Override
/** renvoie de la population*/
 public String toString() {
      return Arrays.toString(individus.toArray());
 }
 /** evaluer une population, */
 public void evaluer(Environnement cible){
                for(int i =0;i<individus.size();i++){
                        individus.get(i).setFitness(cible.eval(individus.get(i)));
                }
                for(int i =0;i<individus.size();i++){
                        for(int j =i+1;j<individus.size();j++){
                                if(individus.get(i).compareTo(individus.get(j))<0){
                                        Collections.swap(individus, i, j);
                                }
                        }
                }
 }
 /**muter la population, avec une valeur en parametre*/
 public void muter(double m){
                for(int i=0; i< individus.size(); i++)
               individus.get(i).setFitness(individus.get(i).getFitness()+individus.get(i).getFitness()*m);
 }
 /**reproduire une population*/
 private Population reproduire(){
                Random r = new Random();
                /** evaluer la population dans l'environnement cible*/
               Population nouvpop =new Population();
               int taille20prcent = individus.size()/5;
               /** recuperation et clonage des 20% individus  */
               for(int i =0; i<taille20prcent;i++){
                        Individu i2 =individus.get(i).clone();
                        nouvpop.add(i2);
               }
               /** reproduction, croiser et muter les 80% individus*/
               for(int j=0; j<individus.size()-taille20prcent; j++){
       Individu i1, i2, i3;

                        i1 = individus.get(r.nextInt(taille20prcent));
                        i2 =individus.get(r.nextInt(taille20prcent));
                        i3=i1.croiser(i2);
```

```
                        i3.muter();
                        nouvpop.add(i3);
                }
                /** renvoie de la nouvelle generation*/
                return nouvpop;
 }
/**evolution d'une population dans un environnement*/
 public IPopulation evoluer(Environnement cible){
                this.evaluer(cible);
                Population pp = reproduire();
                pp.evaluer(cible);
                return pp;
 }

@Override
public Iterator<Individu> iterator() {
                return individus.iterator();
}
public Individu get(int i){
                return individus.get(i);
                }
}
```

```
package pobj.algogen;

public abstract class AbstractIndividu implements Individu {
          private double fitness = 0.0;
          @Override
          public double getFitness(){
                    return fitness;
          }
          @Override
          public void setFitness(double fit) {
                    fitness=fit;
          }
          @Override
          public int compareTo(Individu o){
                    return Double.compare(getFitness(), o.getFitness());
          }
          @Override
          public abstract AbstractIndividu clone();
}
```

package pobj.algogen.adapter.agent :

```
package pobj.algogen.adapter.agent;

import pobj.algogen.Environnement;
import pobj.algogen.Individu;
import agent.Simulation;
import agent.control.IControleur;
import agent.laby.Labyrinthe;

public class LabyEnvironnementAdapter implements Environnement {
private Labyrinthe laby=null;
private int nbSteps =0;
public LabyEnvironnementAdapter(Labyrinthe l, int nb){
                laby=l;
                nbSteps=nb;
}
                @Override
                public double eval(Individu i) {
                                IControleur ctrl= ((ControleurIndividuAdapter)i).getValeurPropre();
                                Simulation s =new Simulation(laby.clone(),ctrl);

                                return s.mesurePerf(nbSteps);
                }

}
```

```
package pobj.algogen.adapter.agent;

import agent.control.ControlFactory;
import agent.control.IControleur;
import pobj.algogen.AbstractIndividu;
import pobj.algogen.Individu;

public class ControleurIndividuAdapter extends AbstractIndividu implements Individu {
                private IControleur controleur;
                public ControleurIndividuAdapter(IControleur control){
                                controleur=control;
                }
                public ControleurIndividuAdapter(int nbRules){
                                controleur=ControlFactory.createControleur( nbRules);
                }
                public String toString(){
                                String cont = ""+ getValeurPropre() + ", " + getFitness();
                                return cont;
                }
                public IControleur getValeurPropre(){
                                return controleur;
                }
                @Override
                public void muter() {
                                getValeurPropre().muter(0.05);
                }
                @Override
```

```
	public Individu croiser(Individu autre) {
		return new ControleurIndividuAdapter(getValeurPropre().creeFils(((ControleurIndividuAdapter)autre).getValeurPropre(), 0.05));
	}
	@Override
	public ControleurIndividuAdapter clone() {
		return new ControleurIndividuAdapter(controleur.clone());
	}
}
*******************************************************************************
package pobj.algogen.adapter.agent;

import agent.laby.Labyrinthe;
import pobj.algogen.Environnement;
import pobj.algogen.IPopulation;
import pobj.algogen.Individu;
import pobj.algogen.Population;


public class PopulationFactory {

	public static IPopulation createRandomPopulation(int size,int nbrules ){
		Population pop=new Population();
		for(int i=0;i<size;i++){
			pop.add(createIndividu(nbrules));
		}
		return pop;
	}

	public static Individu createIndividu(int nbre){
		return new ControleurIndividuAdapter(nbre);
	}
	public static Environnement createEnvironnement(Labyrinthe la, int nbsteps){
		return new LabyEnvironnementAdapter(la,nbsteps);
	}
}
*******************************************************************************
package pobj.algogen.adapter.agent;

import java.io.IOException;
import pobj.algogen.Environnement;
import pobj.algogen.IPopulation;
import agent.laby.ChargeurLabyrinthe;
import agent.laby.Labyrinthe;
public class EvolutionAgentMain {

	/**
	 * @param args
	 */
	public static void main(String[] args) {

				String labyFile = "goal.mze"; // nom du fichier, args[0];
				int taille_pop= 10; // la taille de popolation.
				int nbSteps = 100; // le nbre de pas,Integer.parseInt(args[1]);
				int nbRules = 10; // nbre de regles par controleur, Integer.parseInt(args[2]);
				int nbregeneration =10; // le nbre de genertion pour chaque evaluation.
				Labyrinthe laby = null; // environnement.
				try {
					 laby = ChargeurLabyrinthe.chargerLabyrinthe(labyFile);
				} catch (IOException e) {
					System.out.println("Problème de chargement du labyrinthe"+e);
					System.exit(1);
				}
				System.out.println("j'utilise une population de taille : " + taille_pop);
				IPopulation pop = PopulationFactory.createRandomPopulation(taille_pop, nbRules );
				System.out.println(pop);
				System.out.println("evolution de la population : ");
				Environnement env = PopulationFactory.createEnvironnement(laby, nbSteps);

				for(int i=0;i<nbregeneration;i++){
					System.out.println("evolution de la generation : " +i);
					System.out.println(pop);
					pop.evoluer(env);
				}
				System.out.println("la meilleur Individu de la derniere generation est : " +pop.iterator().next());
	}

}
*******************************************************************************
```

package pobj.algogen.adapter.arith :

```
 package pobj.algogen.adapter.arith;

import pobj.algogen.AbstractIndividu;
import pobj.algogen.Individu;
import pobj.arith.Expression;
import pobj.arith.Expressionfactory;
import pobj.arith.Operator;

public class IndividuExpression extends AbstractIndividu implements Individu {
	private Expression valeurPropre;
	public IndividuExpression(){
		this(Expressionfactory.createRandomExpression());
	}
	public IndividuExpression(Expression e){
		valeurPropre=e;
```

```
	}
	public Expression getValeurPropre(){
		return valeurPropre;
	}
	public String toString(){
		return "valeurpropre =" + getValeurPropre() + ", fitness = " + getFitness();
	}
	public IndividuExpression croiser(Individu ind){
		return new IndividuExpression(Expressionfactory.createOperateurbinaire(Operator.DIV,
				Expressionfactory.createOperateurbinaire(Operator.PLUS, getValeurPropre(),
					((IndividuExpression)ind).getValeurPropre()),Expressionfactory.createConstant(2)).simplifier());
	}

	public void muter(){
		this.valeurPropre=Expressionfactory.createRandomExpression();
	}
	public IndividuExpression clone(){
		return new IndividuExpression(valeurPropre.clone());
	}
}
*********************************************************************************
 package pobj.algogen.adapter.arith;

import pobj.algogen.Environnement;
import pobj.algogen.Individu;
import pobj.arith.EnvEval;
import pobj.arith.Expressionfactory;

public class ValeurCible implements Environnement {
	private double cible;
	private EnvEval env;

	/**constructeur sans parametre*/
	public ValeurCible(){
		cible = Math.random();
		env = Expressionfactory.createRandomEnvironment();
	}

/**evaluer un individu dans un environnement,
 * renvoie la fitness*/
	@Override
	public double eval(Individu i) {
		 double r1= ((IndividuExpression)i).getValeurPropre().eval(env);
		 return 1/((cible -r1)*(cible-r1));
	}
	public String toString(){
		return "objectif f(" +env+")=" + cible;
	}
}
*********************************************************************************
package pobj.algogen.adapter.arith;

import pobj.algogen.Environnement;
import pobj.algogen.IPopulation;
import pobj.algogen.Population;



public class PopulationFactory{

public static IPopulation createRandompopulation(int size){
    Population pop = new Population() ;
        for( int i=0; i<size; i++){
           pop.add (new IndividuExpression());
          }
  return pop;
}
public static Environnement createEnvironnement(){
	return new ValeurCible();
}
}
*********************************************************************************
package pobj.algogen.adapter.arith;

import pobj.algogen.Environnement;
import pobj.algogen.IPopulation;

public class ArithPopulationMain {
	public static void main(String[] args) {

		int taille_pop = 10;
		IPopulation pop = PopulationFactory.createRandompopulation(taille_pop);
		System.out.println("evoluer la population pop ");

		Environnement env= PopulationFactory.createEnvironnement();
		for(int i=0;i<10;i++){
			System.out.println("generation : " + i);
			System.out.println(pop);
			pop=pop.evoluer(env);
		}
		System.out.println();
		System.out.println("resultat pop finale est : " + pop);
	}
}
*****************************************************************************************************************************************************
```

package test.adapter :

```
package test.adapter;

import agent.control.ControlFactory;
import agent.control.IControleur;
import pobj.algogen.Individu;
import pobj.algogen.adapter.agent.ControleurIndividuAdapter;
import junit.framework.TestCase;

public class ControleurIndivAdapterTest extends TestCase {
	ControleurIndividuAdapter cia;
	IControleur controleur;
	Individu autre;

	public ControleurIndivAdapterTest(String arg0) {
		super(arg0);
		controleur = ControlFactory.createControleur(20);
		IControleur controleur2 = ControlFactory.createControleur(20);
		cia = new ControleurIndividuAdapter(controleur);
		autre = new ControleurIndividuAdapter(controleur2);
	}

	protected void setUp() throws Exception {
		super.setUp();
	}

	protected void tearDown() throws Exception {
		super.tearDown();
	}
/*
 * on vérifie que la valeur propre stockée est bien celle qui a été créée initialement
 */
	public void testGetValeurPropre() {
		IControleur cont = cia.getValeurPropre();
		assertTrue(cont.equals(controleur));
		assertTrue(cont==controleur);
	}
/*
 * On vérifie à l'issue du croisement que le nouvel individu n'est pas égal à l'original
 */
	public void testCroiser() {
		cia.croiser(autre);
		assertFalse(cia.equals(autre));
	}
/*
 * On vérifie que la mutation introduit bien un changement
 */
	public void testMuter() {
		autre = cia.clone();
		cia.muter();
		assertFalse(cia.equals(autre));
	}

}
********************************************************************************
/**
 *
 */
package test.adapter;

import java.io.IOException;

import agent.laby.ChargeurLabyrinthe;
import agent.laby.Labyrinthe;
import pobj.algogen.Population;
import pobj.algogen.adapter.agent.PopulationFactory;
import pobj.algogen.adapter.agent.ControleurIndividuAdapter;
import pobj.algogen.adapter.agent.LabyEnvironnementAdapter;
import junit.framework.TestCase;

/**
 * @author sigaud
 *
 */
public class LabyEnvAdapterTest extends TestCase {
	private LabyEnvironnementAdapter env;
	private Labyrinthe labyTest;
	private	Population pop;
	/**
	 * @param arg0
	 */
	public LabyEnvAdapterTest(String arg0) {
		super(arg0);
	}

	/* (non-Javadoc)
	 * @see junit.framework.TestCase#setUp()
	 */
	protected void setUp() throws Exception {
		super.setUp();
		try {
			labyTest = ChargeurLabyrinthe.chargerLabyrinthe("trial.mze");
		} catch (IOException e) {
			e.printStackTrace();
			fail("Could not find test maze !");
		}
		env = new LabyEnvironnementAdapter(labyTest, 10);
		pop = (Population) PopulationFactory.createRandomPopulation(100,20);
```

```
	}

	/* (non-Javadoc)
	 * @see junit.framework.TestCase#tearDown()
	 */
	protected void tearDown() throws Exception {
		super.tearDown();
	}

	/**
	 * Test method for {@link pobj.algogen.adapter.agent.LabyEnvironnementAdapter#eval(pobj.algogen.Individu)}.
	 */
	public void testEval() {
		ControleurIndividuAdapter indiv = (ControleurIndividuAdapter) pop.get(0);
		double retour = env.eval(indiv);
		assert(retour >0);
	}

}
**********************************************************************************************************************************************************
package test.adapter;

import java.io.IOException;

import pobj.algogen.Individu;
import pobj.algogen.Population;
import pobj.algogen.adapter.agent.LabyEnvironnementAdapter;
import pobj.algogen.adapter.agent.PopulationFactory;
import agent.laby.ChargeurLabyrinthe;
import agent.laby.Labyrinthe;
import junit.framework.TestCase;

public class PopulationTest extends TestCase {
	private LabyEnvironnementAdapter env;
	private Labyrinthe labyTest;
	private	Population pop;

	public PopulationTest(String arg0) {
		super(arg0);
	}

	protected void setUp() throws Exception {
		super.setUp();
		try {
			labyTest = ChargeurLabyrinthe.chargerLabyrinthe("trial.mze");
		} catch (IOException e) {
			fail("Could not find test maze !");
		}
		env = new LabyEnvironnementAdapter(labyTest, 10);
		pop = (Population) PopulationFactory.createRandomPopulation(100,20);
		}

	protected void tearDown() throws Exception {
		super.tearDown();
	}

	public void testSize() {
		assertTrue(pop.size()==100);
	}
/*
 * On vérifie que la taille augmente de 1
 */
	public void testAdd() {
		int size = pop.size();
		Individu individu= (pop.get(0)).clone();
		pop.add(individu);
		assertTrue(pop.size()==size+1);
	}
/*
 * on vérifie que la nouvelle population est différente de la précédente
 */
	public void testEvoluer() {
		Population pop2 = (Population) pop.evoluer(env);
		assertTrue(!pop2.equals(pop));
	}
/*
 * On vérifie que les individus sont bien rangés dans l'ordre
 */
	public void testEvaluer() {
		pop.evaluer(env);
		int max = pop.size();
		for (int i = 0; i < max-1; i++) {
			assertTrue(pop.get(i).getFitness()>=pop.get(i+1).getFitness());
		}
	}
}
**********************************************************************************************************************************************************
**********************************************************************************************************************************************************
*
```

TME7

package pobj.obs :

```
package pobj.obs;

public interface ISimpleObservable {
	public void addOvserver(ISimpleObserver o);
```

```
	public void deleteObserver(ISimpleObserver o);
	public void notifyObserver();
}
```

****

```
package pobj.obs;

public interface ISimpleObserver {
	public void update();
}
```

****

```
package pobj.obs;
import java.util.ArrayList;
import java.util.List;

public class SimpleObservable implements ISimpleObservable {
		private List<ISimpleObserver> obs = new ArrayList<ISimpleObserver>();
		@Override
		public void addOvserver(ISimpleObserver o) {
			obs.add(o);
		}

		@Override
		public void deleteObserver(ISimpleObserver o) {
			obs.remove(o);
		}

		@Override
		public void notifyObserver() {
		for(ISimpleObserver o : obs)
			o.update();
		}
}
```

****

package agent :

```
   package agent;

import pobj.obs.SimpleObservable;
import agent.control.IControleur;
import agent.laby.Labyrinthe;

public class Simulationobs extends SimpleObservable {

		/** L'agent qu'on simule */
		private Agent agent;

		/**
		 * Le labyrinthe dans lequel la simulation se déroule.
		 * Attention, ce labyrinthe est modifié au cours de la simulation.
		 */
		private Labyrinthe laby;


		/**
		 * Constructeur : initialise une simulation d'un contrôleur donné dans un labyrinthe donné.
		 * @param m : le labyrinthe, sera modifié par la simulation
		 * @param c : le controleur
		 */
		public Simulationobs(Labyrinthe m, IControleur c) {
			laby = m;
			agent = new Agent(c, m.getPositionInitiale());
		}


		/**
		 * Rend le labyrinthe dans lequel la simulation évolue.
		 * @return mon labyrinthe.
		 */
		public Labyrinthe getLaby() {
			return laby;
		}

		/**
		 * Renvoie le score de l'agent
		 */
		public int getScore() {
			return agent.getScore();
		}

		/**
		 * Effectue nbPas pas de l'agent dans le labyrinthe
		 * @param nbPas : le nombre de pas
		 * @return : le score de l'agent
		 */
		public int mesurePerf (int nbPas) {
			for (int i=0;i<nbPas;i++){
				agent.faitUnPas(getLaby());
				// mise a jour de observer
				this.notifyObserver();
			}
			return agent.getScore();
		}
		public void setLaby(Labyrinthe l){
			laby = l;
		}
		public Agent getAgent(){
			return agent;
		}
```

```
}
**********************************************************************************
package agent.laby.interf :
  package agent.laby.interf;

import java.awt.event.ActionEvent;

import agent.laby.Labyrinthe;
import pobj.obs.ISimpleObserver;

public class LabyActivePanel extends LabyPanel implements ISimpleObserver {

	/**
	 *
	 */
	private static final long serialVersionUID = 1L;

	public LabyActivePanel(Labyrinthe laby) {
		super(laby);
	}
	@Override
	public void update() {
		updateGraphics();
		try{
			Thread.sleep(500);
		}catch(InterruptedException e){
			e.printStackTrace();
		}
	}
	public void actionPerformed(ActionEvent e) {

	}
}
**********************************************************************************
  package agent.laby.interf;

import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class Configuration extends JPanel {
	/**
	 *
	 */
	private static final long serialVersionUID = 1L;
	JTextField taillePopC, nbrePasC, nbregeneraC;
	JLabel taillepoplabel, nbregeneratlabe, nbrepaslabel, valider;
	private JButton b;

	public Configuration(){
		this.setLayout(new GridLayout(4,1));
		 //ajouter la taille de la population.
		  taillePopC = new JTextField();
		 taillepoplabel = new JLabel("Taille Pop");
		 this.add(taillepoplabel);
		 this.add(taillePopC);
		 //ajouter le nbre de pas
		 nbrePasC =new JTextField();
		 nbrepaslabel = new JLabel("PAS");
		 this.add( nbrepaslabel);
		 this.add(nbrePasC);
		 //ajouter le nbre de generation.
		 nbregeneraC =new JTextField();
		 nbregeneratlabe = new JLabel("Nbre Gen");
		 this.add( nbregeneratlabe);
		 this.add( nbregeneraC);
		 valider =new JLabel("valider");
		 //ajouter un bouton pour valider les données
		 b=new JButton("ok");
		 this.add(valider);
		 this.add(b);
	}


	public JButton getButon(){return b;}
	public JTextField getTaillePop(){return taillePopC;}
	public JTextField getNbreGenera(){return nbregeneraC;}
	public JTextField getNbrePas(){return nbrePasC;}


}
**********************************************************************************
package agent.laby.interf;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
```

```
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.WindowConstants;
import javax.swing.filechooser.FileNameExtensionFilter;

import pobj.algogen.Environnement;
import pobj.algogen.IPopulation;
import pobj.algogen.Individu;
import pobj.algogen.Population;
import pobj.algogen.adapter.agent.AgentEnvironnementAdapter;
import pobj.algogen.adapter.agent.ControleurIndividuAdapter;
import pobj.algogen.adapter.agent.PopulationFactory;
import agent.Simulationobs;
import agent.control.IControleur;
import agent.laby.ChargeurLabyrinthe;
import agent.laby.ContenuCase;
import agent.laby.Labyrinthe;


public class LabyViewer extends JFrame {
	private static final long serialVersionUID = 1L;

	private JPanel sidePanel;
	public  Configuration configuration  ;
	private IControleur controleur ;
	private Labyrinthe laby;
	private LabyActivePanel centerPanel;
	//private static final int COLS = 15, LIGNES = 10;
	int nbRules = 10;
	int nbPas, taille_pop, nbgeneration;
	IPopulation pop ;
	Environnement envir;
	/**

 * Constructeur
	 */
	public LabyViewer(Labyrinthe lab, IControleur ctrl, int nb) {
		super("Laby Viewer");
		//laby = new Labyrinthe(COLS, LIGNES);
		laby = lab;
		controleur = ctrl;
		nbPas=nb;
		configuration = new Configuration();
		createCenterPanel();
		createSidePanel();
		setSize(800, 658);
		setResizable(false);
		setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
		setVisible(true);
	}



	/**
	 * Crée le panneau latéral, ses boutons et associe les actions appropriées aux boutons.
	 */
	public void createSidePanel() {
		sidePanel = new JPanel();
		//sidePanel.setLayout(new BoxLayout(sidePanel, BoxLayout.Y_AXIS));
		sidePanel.setLayout(new GridLayout(3,1));

		JTextArea instructions = new JTextArea();
		instructions.setText("Cliquez sur play pour\n lancer le jeu\n de labyrinthe!\n");
		instructions.setEditable(false);
		sidePanel.add(instructions);

		configuration.getButon().addActionListener(new ActionListener(){
					public void actionPerformed(ActionEvent e) {
						try{

						 setPas(Integer.parseInt((configuration.getNbrePas()).getText()));
						 taille_pop=Integer.parseInt((configuration.getTaillePop()).getText());
						 nbgeneration=Integer.parseInt((configuration.getNbreGenera()).getText());


						}catch(NumberFormatException e1){
							System.out.println("je veux des entiers merci.");
						}

						pop =(Population) PopulationFactory.createRandomPopulation(taille_pop, nbRules);

						System.out.println("evolution de la population : ");

						Environnement env = PopulationFactory.createEnvironnement(laby, nbPas);
						for(int i=0;i< nbgeneration;i++){
							pop =(Population) pop.evoluer(env);
						}
						envir = new AgentEnvironnementAdapter(laby,nbPas);

						//recuperer le meilleur controleur
						int meilleurscore=0;
						for (int i = 0; i< taille_pop;i++){
							Individu in = ((Population)pop).get(i);
							if(envir.eval(in)>meilleurscore){
								meilleurscore=(int) envir.eval(in);
								controleur = ((ControleurIndividuAdapter)((Population)pop).get(i)).getValeurPropre();
							}
```

```
                                        }
                                        setControl(controleur);
                                        System.out.println("le meilleur Controleur est : "+controleur);
                                        System.out.println("la taille de la population est : "+taille_pop);
                                        System.out.println("le meilleur score est : "+meilleurscore);

                                }
                });

                sidePanel.add(configuration);

                JButton jouer = new JButton("Play");
                jouer.setIcon(null);
                sidePanel.add(jouer);


                jouer.addActionListener(new ActionListener() {
                        @Override
                        public void actionPerformed(ActionEvent e) {
                                new Thread(new Runnable(){
                                        public void run(){
                                                Labyrinthe lab= laby.clone();
                                                System.out.println(lab);
                                                Simulationobs sim = new Simulationobs(lab,controleur);
                                                centerPanel.setLaby(sim.getLaby());
                                                sim.addOvserver(centerPanel);
                                                sim.mesurePerf(getnbPas());
                                                System.out.println(sim.getLaby());
                                        }

                                }).start();
                        }
                });

                getContentPane().add(sidePanel, BorderLayout.EAST);
}

public void setControl(IControleur d){controleur=d;}
public void setPas(int a ){nbPas=a;}
public int getnbPas(){return nbPas;}


/**
 * Crée le MazePanel responsable d'afficher le Maze courant.
 */

private void createCenterPanel() {
                centerPanel = new LabyActivePanel(laby);
                getContentPane().add(centerPanel, BorderLayout.CENTER);

}

/**
 * Export du labyrinthe par la sérialisation
 *
 * @throws IOException
 */
public void exportMazeData() throws IOException {
                // Force la remise à jour de l'état du labyrinthe en fonction des boutons
                // affichés dans l'interface graphique de dessin
                // What You See Is What You Get
                centerPanel.modifLaby();

                String fileName = JOptionPane
                                .showInputDialog("Please enter a file name to save this maze (extension .mze).");
                ChargeurLabyrinthe.sauverLabyrinthe(fileName, laby);
}

/**
 * Import du labyrinthe sauvé par la sérialisation
 *
 * @throws IOException
 * @throws ClassNotFoundException
 */
public void chargerLabyrinthe() throws IOException {

                // / Code pris directement dans la doc de JFileChooser.
                // L'argument "./" permet de démarrer directement dans le repertoire
                // courant
                // La version par défaut JFilechooser() démarre dans le $home.
                JFileChooser chooser = new JFileChooser(new File("./"));
                FileNameExtensionFilter filter = new FileNameExtensionFilter("Maze files", "mze");
                chooser.setFileFilter(filter);
                chooser.setDialogTitle("Entrez un nom de fichier .mze (avec l'extension");
                int returnVal = chooser.showOpenDialog(this);
                String fileName;
                if (returnVal == JFileChooser.APPROVE_OPTION) {
                        fileName = chooser.getSelectedFile().getName();
                } else {

                        return;
                }

                laby = ChargeurLabyrinthe.chargerLabyrinthe(fileName);
                centerPanel.setLaby(laby);
}
```

```
	/**
	 * Export du labyrinthe au format xml
	 *
	 * @throws IOException
	 */
	public void sauverLabyrintheEnXML() throws IOException {
		centerPanel.modifLaby();
		ChargeurLabyrinthe.sauverLabyrintheEnXML("maze_xml.txt", laby);
	}

	/**
	 * Getter
	 *
	 * @return : le labyrinthe
	 */
	public Labyrinthe getLaby() {
		return laby;
	}

	/**
	 * Méthode principale
	 *
	 * @param args
	 *		: non utilisé
	 */
	public static void main(String[] args) {

		String labyFile = "goal.mze"; // nom du fichier, args[0];
		//int taille_pop= 500; // la taille de popolation.
		int nbPas = 0; // le nbre de pas,Integer.parseInt(args[1]);
		Labyrinthe laby =null; // environnement.
		IControleur contr=null; //controleur.

		try {
			laby = ChargeurLabyrinthe.chargerLabyrinthe(labyFile);
			laby.setContenuCase(laby.getPositionInitiale().x,laby.getPositionInitiale().y, ContenuCase.AGENT);
		} catch (IOException e) {
			System.out.println("Problème de chargement du labyrinthe"+e);
			System.exit(1);
		}

		new LabyViewer(laby, contr, nbPas);

		System.out.println("----------------------------");
		System.out.println("le labyrinthe apres simulation Observer");
		System.out.println(laby.toString());
	}
}
```

****

TME8

dans le package pobj.util :

```
 package pobj.util;

public interface Environnement {
	double valeurCible = Math.random();
	public double eval(Individu i);
}
```

****

```
 package pobj.util;

public class ValeurCible implements Environnement {
	private double value;
/**constructeur sans parametre*/
	public ValeurCible(){
		value = Math.random();
	}
	/** constructeur avec parametre*/
	public ValeurCible(double val){
		value = val;
	}
/**evaluer un individu dans un environnement, renvoie la fitness*/
	@Override
	public double eval(Individu i) {
		 i.setFitness(1/(Math.pow(Environnement.valeurCible -i.getValeurPropre(),2)));
		 return i.getfitness();
	}
	/**accesseur*/
	public double getValue(){return value;}
	/**modificateur*/
	public void setValue(double vale){value = vale;}

	public String toString(){
		return "aa";
	}

}
```

****

```
 package pobj.util;
import java.util.*;

public class Individu implements Comparable<Individu>{
```

```
/**valeur propre pour chaque individu*/
        private double valeurPropre;

        private double fitness = 0;
/** valeur propre aleatoire de individu*/
        public Individu(){
                Random r = new Random();
                valeurPropre = r.nextInt(100);
        }
/** constructeur avec parametre*/
        public Individu(double in){
                valeurPropre=in;
        }
/** accesseur*/
        public double getValeurPropre(){return valeurPropre;}

/** modificateur*/
        public void setValeurPropre(double valeur){
                   valeurPropre= valeur;
        }
/**accesseur*/
        public double getfitness(){return fitness;}
/**modificateur*/
        public void setFitness(double fit){
         fitness = fit;
        }
/** methode toString de individu*/
        public String toString(){
         return "[ "+ valeurPropre +", " + fitness + "]";
        }
/**methode compareTo*/
        public int compareTo(Individu o){
                if(this.getfitness()>o.getfitness()){
                return 1;
                }else if(this.getfitness()<o.getfitness()){
                        return -1;
                        }else{
                                return 0;
                        }
        }
/**croiser l'objet courant, l individu, avec l objet (individu) passé en parametre */
        public Individu croiser(Individu autre){
                Individu nouvelIndividu = new Individu((this.getValeurPropre()+autre.getValeurPropre())/2);
                nouvelIndividu.fitness=this.getfitness();
                return nouvelIndividu;
        }
/** muter l'objet courant avec 5%, changer sa valeur propre, et garder la meme fitness*/
        public void muter(){
                this.valeurPropre = getValeurPropre() * 1.05;
                this.fitness = getfitness();
        }
/**cloner un individu, j ai choisi juste de doublé ses valeurs.*/
        public Individu clone(){

                return new Individu(valeurPropre) ;
                }
}
```

****

```
  package pobj.util;
import java.util.Arrays;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class Population {

  /** liste d individu*/
 // ArrayList<Individu> individus;
 MyArrayList<Individu> individus;
  /** la taille de la population*/
  @SuppressWarnings("unused")
private int size = 0;

  /** instancier une population*/
 public Population() {
       // individus = new ArrayList<Individu>();
                                        individus= new MyArrayList<Individu>(5);
 }

  public int size () {
        //return size;
        return individus.size();
  }
 /**ajouter un individu*/
  public void add (Individu individu) {
       individus.add(individu);
  }
  @Override
/** renvoie de la population*/
  public String toString() {
        //return Arrays.toString(individus.toArray());
                return individus.toString();
  }
/** methode supplementaire, juste pour mieux voir l'evaluation sans tri*/
  public void evaluer_sans_tri(Environnement cible){
                for(int i =0;i<individus.size();i++){
                        individus.get(i).setFitness(cible.eval(individus.get(i)));
```

```
		}
	}
	/** evaluer une population, */
	public void evaluer(Environnement cible){
		for(int i =0;i<individus.size();i++){
			individus.get(i).setFitness(cible.eval(individus.get(i)));
		}
		for(int i =0;i<individus.size();i++){
			for(int j =i+1;j<individus.size();j++){
				if(individus.get(i).compareTo(individus.get(j))<0){
					Collections.swap(individus, i, j);
				}
			}
		}
	}
	/**muter la population, avec une valeur en parametre*/
	@SuppressWarnings("unused")
	private void muter(double m){
		for(int i=0; i< individus.size(); i++)
		((Individu) individus.get(i)).setFitness(((Individu) individus.get(i)).getfitness()+((Individu) individus.get(i)).getfitness()*m);
	}
	/**reproduire une population*/
	private Population reproduire(){
		/** evaluer la population dans l'environnement cible*/
		Population nouvpop =new Population();
		int taille20prcent = individus.size()/5;
		/** recuperation et clonage des 20% individus  */
		for(int i =0; i<taille20prcent;i++){
			Individu i2 =((Individu) individus.get(i)).clone();
			nouvpop.add(i2);
		}
		/** reproduction, croiser et muter les 80% individus*/
		for(int j=0; j<individus.size()-taille20prcent; j++){
	Individu i1, i2, i3;
			i1 = (Individu) individus.get(new Random().nextInt(taille20prcent));
			i2 =(Individu) individus.get(new Random().nextInt(taille20prcent));
			i3=i1.croiser(i2);
			i3.muter();
			nouvpop.add(i3);
		}
		/** renvoie de la nouvelle generation*/
		return nouvpop;
	}
	/**evolution d'une population dans un environnement*/
	public Population evoluer(Environnement cible){
		this.evaluer(cible);
		Population pp = new Population();
		pp = reproduire();
		return pp;
	}
}
```

```
package pobj.util;


public class PopulationFactory{

public static Population createRandompopulation(int size){
		Population pop = new Population() ;
			for( int i=0; i<size; i++){
			    pop.add (new Individu());
			   }
  return pop;
}
}
```

```
package pobj.util;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Vector;

public class MyArrayList<T extends Comparable<T>> extends java.util.AbstractList<T> implements Iterable<T>{

	private static int tailleMAX =5;

	private LinkedList<Vector<T>> list;

	public  MyArrayList(){
		list= new LinkedList< Vector<T> >();
	}

	public MyArrayList(int t){
		tailleMAX = t;
		//Vector<T> v = new Vector<T>(tailleMAX);

		list = new LinkedList<Vector<T>>();
	}
	@SuppressWarnings("unchecked")
	public MyArrayList(Collection<T> c){
		list = new LinkedList<Vector<T>>();
		for(int i=0; i< c.toArray().length; i++){
			this.add((T)c.toArray()[i]);

		}
```

```
	}
	/**
	 * ajout d'un objet de type T a la liste
	 */
	public boolean add(T object){
		if(list.isEmpty()){
			Vector<T> v = new Vector<T>(tailleMAX);
			v.add(object);
			return list.add(v);
		}else if(list.getLast().size()==tailleMAX){
				Vector<T> v = new Vector<T>(tailleMAX);
				v.add(object);
				return list.add(v);
			}else{
				return list.getLast().add(object);
			}
		}

	/**renvoie de l'element a la position location dans la liste*/
	public T get(int location){
		return list.get(location/tailleMAX).get(location % tailleMAX) ;
	}
	/**remplacer l'objet a la position location pas l'objet passé en parametre*/
	public T set(int location, T object){
		return list.get(location/tailleMAX).set(location % tailleMAX, object) ;
	}
	/**renvoie la taille de la liste chainée*/
	public int size(){
		int size=0;
		for(int i=0; i<list.size(); i++){
			size+=list.get(i).size();
		}
		return size;
	}
	/**accesseur, renvoie la liste chainée*/
	public LinkedList<Vector<T>> getList() {
		return list;
	}
	/**modficateur, remplacer la liste par la liste passé en parametre*/
	public void setList(LinkedList<Vector<T>> list) {
		this.list = list;
	}
	/**renvoie la chaine de caractere de la liste*/
	public String toString(){
		return list.toString();
	}
	/**un iterator sur la MyArrayList*/
	@Override
	public Iterator<T> iterator() {
		// TODO Auto-generated method stub
		return new MyIterator<T>(list);

	}
}
	/*
	 * la classe MyIterator definie les deux iterateurs de MyArrayList, un iterateur sur la liste,
	 * et un autre iterateur sur chaque elements de la liste, c est a dire sur les elements de chaque Vector
	 */
	class MyIterator<T> implements Iterator<T>{   //implements ListIterator<T>
	/**iterateur sur la liste globale de MyArrayList*/
		private Iterator<Vector<T>> listIT ;
	/**iterateur sur chaque vecteur local de la liste*/
		private Iterator<T> vectIT ;

		public MyIterator(List<Vector<T>> it){
			listIT = (Iterator<Vector<T>>)it.iterator();
			vectIT = (Iterator<T>) Collections.EMPTY_LIST.iterator();
		}

		@Override
		public boolean hasNext() {
			// TODO Auto-generated method stub
			return (listIT.hasNext() || vectIT.hasNext());
		}
		@Override
		public T next() {
			if( !vectIT.hasNext()){
				vectIT=listIT.next().iterator();
			}else {
				return vectIT.next();
			}
			 return vectIT.next();
		}




		@Override
		public void remove() throws UnsupportedOperationException {
			// TODO Auto-generated method stub
		}


	public Iterator<Vector<T>> getListIT() {
		return listIT;
	}
```

```
	public void setListIT(Iterator<Vector<T>> listIT) {
		this.listIT = listIT;
	}

	public Iterator<T> getVectIT() {
		return vectIT;
	}
	public void setVectIT(Iterator<T> vectIT) {
		this.vectIT = vectIT;
	}

}
********************************************************************************
package pobj.util;

import java.util.Iterator;

public class Test {

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		// TODO Auto-generated method stub
		System.out.println("hello");
		MyArrayList<Integer> maliste= new MyArrayList<Integer>(5);

		for(int i= 0; i<=23;i++){
		maliste.add(i);
		}
		System.out.println(maliste.toString());
		System.out.println(maliste.get(10));
		System.out.println(maliste.size());
		maliste.set(20, 50);
		System.out.println(maliste.toString());
		maliste.set(13, 333333);
		System.out.println(maliste.toString());

		for(Integer a : maliste){
				System.out.println(a);
		}
		for(Iterator<Integer> it = maliste.iterator(); it.hasNext();){
			System.out.print(it.next() + " ");
		}
	}

}
********************************************************************************
 package pobj.util;

public class PopulationMain{

	public static void main (String[] args){

	System.out.println("-------------------------TME8-------------------------------------------");

	Chrono time5_TME8 = new Chrono();       // pour le temps total
	Chrono time1_TME8 = new Chrono();         //  le temps de la creation (declaration + allocation) d'une linkedlist
	Population popu = new Population();
	popu =  PopulationFactory.createRandompopulation(500);
	System.out.print("le temps necessaire pour la creation (declartion + allocation) d'une population avec une Linkedlist
est :  ");
	time1_TME8.stop();

	Chrono time2_TME8 = new Chrono();     // le temps necessaire pour l'afficahge d'une population avec une linkedlist
	System.out.println("la population aleatoire est : " + popu);
	System.out.print("le temps necessaire pour l'affichage d'une population avec une Linkedlist est :  ");
	time2_TME8.stop();

	System.out.println("la valeur cible de l environnement est : " + Environnement.valeurCible);
	Environnement env = new ValeurCible();
	System.out.println("");
	System.out.println("---------evaluation  + tri decroissant + evolution (10 generations) d'une population crée avec une
MyrrayList LinkedList (liste chainée)--------------");

	Chrono time3_TME8 = new Chrono(); // le temps necessaire pour evaluer tri decroissant et evoluer une population crée
avec une linkedlist
	popu.evaluer(env);
	for (int i =1; i <=10; i++){
	popu = popu.evoluer(env);
	}
	System.out.print("le temps necessaire pour evaluer, tri decroissant et evoluer une population avec une Linkedlist est :
");
	time3_TME8.stop();
	System.out.println(popu.toString());
	System.out.print("le temps final d'une population avec une Linkedlist est :  ");
	time5_TME8.stop();


	}
}
********************************************************************************************************************************************************
**************************
```

TME9

dans le package pobj.util :

```
package pobj.util;

public interface AlgoGenParameter {

	public static final String TAILLE_POP = "taille_population";
	public static final String NB_PAS = "nbpas";
	public static final String NB_RULES = "nbregles";
	public static final String NB_GENES = "nbgen";
}
```

****

```
package pobj.util;

import java.util.Random;

public class Generateur {
	private static Generateur generateur;
	private Random random;
	@SuppressWarnings("unused")
	private long graine;

	public Generateur(long graine){
		this.graine=graine;
		random= new Random(graine);
		random.setSeed(graine);
	}

	public static Generateur getGenerateur(long graine){
		if(generateur==null){
				generateur= new Generateur(graine);
		}
		return generateur;
	}
	public void setSeed(long a){random.setSeed(a);}
	public double nextDouble(){ return random.nextDouble();}
	public int nextInt(){return random.nextInt();}
	public Random getRandom(){
		return random;
	}

}
```

****

```
package pobj.util;

import java.io.BufferedReader;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.util.HashMap;
import java.util.Map;



public class Configuration implements AlgoGenParameter {
	Map<String,String> map;
	private static Configuration instance;


	private Configuration(String fileName){
		map = new HashMap<String, String>();
		try {
			chargerParameters(fileName);
		} catch (IOException e) {
			System.out.println("Probleme au chargement des parametres");
			e.printStackTrace();
		}
	}

	public static Configuration getInstance(String fileName){
		if (instance == null)
			instance = new Configuration(fileName);
		return instance;
	}

	public String getParameterValue(String parameter){
		return map.get(parameter);
	}

	public void setParameterValue(String parameter,String value){
		map.put(parameter, value);
	}

	public void chargerParameters(String file) throws IOException{
		BufferedReader fileIn = new BufferedReader(new FileReader(file));
		String []buffer;
		String line;
		while(fileIn.ready()){
			line = fileIn.readLine();
			buffer = line.split(":");
			String parameter = buffer[0];//le parametre
```

```
                                        String value = buffer[1];//la valeur associ?e
                                        if (parameter.equals(TAILLE_POP))
                                                setParameterValue(parameter, value);
                                        if (parameter.equals(NB_PAS))
                                                setParameterValue(parameter, value);
                                        if (parameter.equals(NB_RULES))
                                                setParameterValue(parameter, value);
                                        if (parameter.equals(NB_GENES))
                                                setParameterValue(parameter, value);
                                }
                        }

                        public static void sauverParameters(String fileOut,Configuration confi) throws IOException{
                                FileOutputStream fos = new FileOutputStream(fileOut);
                                ObjectOutputStream oos = new ObjectOutputStream(fos);
                                oos.writeObject(confi);
                                oos.close();

                        }

}
```

****

dans le package agent.laby.interf :

```
   package agent.laby.interf;

import java.awt.GridLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

public class NouvelJpanel extends JPanel {
                /**
                 *
                 */
                private static final long serialVersionUID = 1L;
                JTextField taillePopC, nbrePasC, nbregeneraC;
                JLabel taillepoplabel, nbregeneratlabe, nbrepaslabel, valider;
                private JButton b;

                public NouvelJpanel(){
                                this.setLayout(new GridLayout(4,1));
                                 //ajouter la taille de la population.
                                  taillePopC = new JTextField();
                                 taillepoplabel = new JLabel("Taille Pop");
                                 this.add(taillepoplabel);
                                 this.add(taillePopC);
                                 //ajouter le nbre de pas
                                 nbrePasC =new JTextField();
                                 nbrepaslabel = new JLabel("PAS");
                                 this.add( nbrepaslabel);
                                 this.add(nbrePasC);
                                 //ajouter le nbre de generation.
                                 nbregeneraC =new JTextField();
                                 nbregeneratlabe = new JLabel("Nbre Gen");
                                 this.add( nbregeneratlabe);
                                 this.add( nbregeneraC);
                                 valider =new JLabel("valider");
                                 //ajouter un bouton pour valider les données
                                 b=new JButton("ok");
                                 this.add(valider);
                                 this.add(b);
                }
                public JButton getButon(){return b;}
                public JTextField getTaillePop(){return taillePopC;}
                public JTextField getNbreGenera(){return nbregeneraC;}
                public JTextField getNbrePas(){return nbrePasC;}
}
```

****

```
  package agent.laby.interf;

import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.IOException;

import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextArea;
import javax.swing.WindowConstants;
import javax.swing.filechooser.FileNameExtensionFilter;

import pobj.algogen.Environnement;
import pobj.algogen.IPopulation;
import pobj.algogen.Individu;
import pobj.algogen.Population;
import pobj.algogen.adapter.agent.AgentEnvironnementAdapter;
import pobj.algogen.adapter.agent.ControleurIndividuAdapter;
import pobj.algogen.adapter.agent.PopulationFactory;
import pobj.util.AlgoGenParameter;
import pobj.util.Chrono;
```

```
import pobj.util.Configuration;
import agent.Simulationobs;
import agent.control.IControleur;
import agent.laby.ChargeurLabyrinthe;
import agent.laby.ContenuCase;
import agent.laby.Labyrinthe;


public class LabyViewer extends JFrame {
	private static final long serialVersionUID = 1L;

	private JPanel sidePanel;
	private IControleur controleur ;
	private Labyrinthe laby;
	private LabyActivePanel centerPanel;
	IPopulation pop ;
	IControleur contr=null;
	Environnement envir;
	int nbPas;
	/**

 * Constructeur
	 * @throws IOException
	 */
	public LabyViewer(int nbpas, int taille_pop, int nbrules,int nbgeneration) throws IOException {
			super("Laby Viewer");
	nbPas=nbpas;
			laby = ChargeurLabyrinthe.chargerLabyrinthe("goal.mze");
	   laby.setContenuCase(laby.getPositionInitiale().x,laby.getPositionInitiale().y, ContenuCase.AGENT);
			System.out.println(laby.toString());
			envir = new AgentEnvironnementAdapter(laby,nbpas);
			pop =(Population) PopulationFactory.createRandomPopulation(taille_pop, nbrules);
			 meilleurAgent( nbgeneration, taille_pop);
			 controleur = contr;
			 createCenterPanel();
			createSidePanel();
			setSize(800, 658);
			setResizable(false);
			setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
			setVisible(true);
	}


	/**
	 * Crée le panneau latéral, ses boutons et associe les actions appropriées aux boutons.
	 */
	public void createSidePanel() {
			sidePanel = new JPanel();
			sidePanel.setLayout(new GridLayout(3,1));

			JTextArea instructions = new JTextArea();
			instructions.setText("Cliquez sur play pour\n lancer le jeu\n de labyrinthe!\n");
			instructions.setEditable(false);
			sidePanel.add(instructions);
			JButton jouer = new JButton("Play");
			jouer.setIcon(null);
			sidePanel.add(jouer);


			jouer.addActionListener(new ActionListener() {
				@Override
				public void actionPerformed(ActionEvent e) {
					new Thread(new Runnable(){
						public void run(){
							Labyrinthe lab= laby.clone();
							System.out.println(lab);
							Simulationobs sim = new Simulationobs(lab,controleur);
							centerPanel.setLaby(sim.getLaby());
							sim.addOvserver(centerPanel);
							sim.mesurePerf(getnbPas());
							System.out.println(sim.getLaby());
						}

					}).start();
				}
			});

			getContentPane().add(sidePanel, BorderLayout.EAST);
	}
	public void meilleurAgent(int nbGen,int taille){
			for(int i=0;i< nbGen;i++){
				pop =(Population) pop.evoluer(envir);
			}
			//recuperer le meilleur controleur
			int meilleurscore=0;
			for (int i = 0; i< taille;i++){
				Individu in = ((Population)pop).get(i);
				if(envir.eval(in)>meilleurscore){
					 meilleurscore=(int) envir.eval(in);
					 contr = ((ControleurIndividuAdapter)((Population)pop).get(i)).getValeurPropre();
				}
			}
			setControl(contr);
	}

	public void setControl(IControleur d){controleur=contr;}
	public void setPas(int a ){nbPas=a;}
```

```
public int getnbPas(){return nbPas;}


/**
 * Crée le MazePanel responsable d'afficher le Maze courant.
 */

private void createCenterPanel() {
		centerPanel = new LabyActivePanel(laby);
		getContentPane().add(centerPanel, BorderLayout.CENTER);

}

/**
 * Export du labyrinthe par la sérialisation
 *
 * @throws IOException
 */
public void exportMazeData() throws IOException {
		// Force la remise à jour de l'état du labyrinthe en fonction des boutons
		// affichés dans l'interface graphique de dessin
		// What You See Is What You Get
		centerPanel.modifLaby();

		String fileName = JOptionPane
						.showInputDialog("Please enter a file name to save this maze (extension .mze).");
		ChargeurLabyrinthe.sauverLabyrinthe(fileName, laby);
}

/**
 * Import du labyrinthe sauvé par la sérialisation
 *
 * @throws IOException
 * @throws ClassNotFoundException
 */
public void chargerLabyrinthe() throws IOException {

		// / Code pris directement dans la doc de JFileChooser.
		// L'argument "./" permet de démarrer directement dans le repertoire
		// courant
		// La version par défaut JFilechooser() démarre dans le $home.
		JFileChooser chooser = new JFileChooser(new File("./"));
		FileNameExtensionFilter filter = new FileNameExtensionFilter("Maze files", "mze");
		chooser.setFileFilter(filter);
		chooser.setDialogTitle("Entrez un nom de fichier .mze (avec l'extension");
		int returnVal = chooser.showOpenDialog(this);
		String fileName;
		if (returnVal == JFileChooser.APPROVE_OPTION) {
			fileName = chooser.getSelectedFile().getName();
		} else {

			return;
		}

		laby = ChargeurLabyrinthe.chargerLabyrinthe(fileName);
		centerPanel.setLaby(laby);
}

/**
 * Export du labyrinthe au format xml
 *
 * @throws IOException
 */
public void sauverLabyrintheEnXML() throws IOException {
		centerPanel.modifLaby();
		ChargeurLabyrinthe.sauverLabyrintheEnXML("maze_xml.txt", laby);
}
/**
 * Getter
 *
 * @return : le labyrinthe
 */
public Labyrinthe getLaby() {
		return laby;
}
public static void importParameter(Configuration configuration){
		int tailleP = 0;
		int nbpas=0;
		int nbregles=0;
		int nbgen=0;

		tailleP = Integer.parseInt(configuration.getParameterValue(AlgoGenParameter.TAILLE_POP));
		nbpas = Integer.parseInt(configuration.getParameterValue(AlgoGenParameter.NB_PAS));
		nbregles = Integer.parseInt(configuration.getParameterValue(AlgoGenParameter.NB_RULES));
		nbgen = Integer.parseInt(configuration.getParameterValue(AlgoGenParameter.NB_GENES));

		try {
			new LabyViewer(nbpas,tailleP,nbregles,nbgen);
		} catch (IOException e) {
			System.out.println("Probleme de chargement du labyrinthe");
			e.printStackTrace();
		}
}
public static void main(String[] args) {
		Chrono chrono = new Chrono();
		Configuration config = Configuration.getInstance("foufa.txt");
		importParameter(config);
```

```
                chrono.stop() ;
        }
}
```
**********FIN**********