ESIAL
Université de Lorraine

Rapport de PIDR

Encadrants universitaires : Martin Quinson & Gérald Oster

Java Learning Machine

Marion Le Bras

Nancy, le 21 mai 2012

Table des matières

1 Introduction

Le *Projet Interdisciplinaire ou Découverte de la Recherche* est un projet de 5 mois servant d'introduction aux élèves de deuxième année à l'univers de la recherche. Dans ce module, contrairement aux projets standards, les élèves sont confrontés à un problème pour lequel il n'existe pas encore de solution explicite. Ils doivent donc explorer des pistes afin de trouver la bonne approche pour résoudre efficacement le problème.

1.1 Contexte

Pour nous, ce projet a consisté à contribuer à l'amélioration d'une plateforme d'aide pour l'apprentissage de la programmation en JAVA, JLM -JAVA Learning Machine. Martin Quinson et Gérald Oster, les développeurs de l'application, ont supervisé ce PIDR.

Java Learning Machine (JLM) est un outil pédagogique développé par Martin Quinson et Gérald Oster permettant aux étudiants d'apprendre les concepts de base de la programmation. Il est utilisé à l'ESIAL dans les modules PPP (Premier Pas en Programmation) et TOP (Techniques et Outils pour Programmer) enseignés en première année. La plate-forme est composée de nombreux exercices à difficulté progressive, couvrant l'ensemble des notions de bases de la programmation (introduction aux variables de données, aux instructions de boucle, aux déclarations et appels de méthodes, etc).

Les exercices sont construits de la manière suivante : une introduction avec un petit scénario, une leçon, et un objectif pour mettre en pratique ce qui a été appris. L'utilisateur acquiert donc pas à pas de nouvelles notions. La plateforme sauvegarde le code tapé par l'étudiant, lui permettant ainsi de revoir ce qu'il a fait, et potentiellement améliorer ce qu'il a fait.

Ainsi, JLM introduit les élèves à des programmations de type procédural et impératif, ce qui constitue déjà une partie importante de la programmation. Cependant l'outil ne possède ni leçons ni exercices sur la programmation orientée objet.

1.2 Objectifs

Notre sujet de PIDR s'inscrit dans la continuité du travail déjà réalisé sur l'application. Nous devons étudier les possibilités d'utilisation de la JLM pour l'apprentissage de la programmation orientée objet - POO. Toutefois nous pouvons nous appuyer sur les travaux du groupe de PIDR des années précédentes de Loïc Vourc'h et Sébastien Toni ainsi que Sébastien Champagne et Gabriel Larroque. Il ont travaillé sur le même sujet et ont analysé les techniques actuelles d'enseignement de la POO.

À partir de cette base, nous avons donc établi notre propre suite d'exercices et les avons déployés sur JLM.

1.3 Enjeux

Les enjeux de ce projet sont multiples. Dans un premier temps les recherches menées doivent conduire à la réalisation de leçons et exercices qui seront utilisés par les élèves utilisant Java

Learning Machine. Il en va donc de l'apprentissage d'élèves ingénieurs dans les langages objet. Dans un deuxième temps, les conclusions de la recherche doivent apporter une solution technique permettant l'implémentation d'autres exercices dans la suite du développement de la plate-forme. Ainsi mis en place, les nouveaux exercices pourront profiter aux futurs étudiants de l'ESIAL.

2 Organisation

2.1 Outils

Afin de continuer à développer JLM sans toucher à sa version stable, nous avons utilisé le gestionnaire de version décentralisé Git avec l'interface GitHub. Ce gestionnaire de version a la propriété de laisser ses utilisateurs "dupliquer" un dépôt, ce que nous avons fait.

Le dépôt sur lequel nous avons travaillé se trouve à l'adresse :

`http://github.com/marionlb/JLM`

.

2.2 Répartition des tâches

Le travail s'est tout d'abord fait en commun, puis nous avons équitablement rétabli les tâches. L'un s'occupait de l'implémentation des leçons, l'autre de l'interface et de la documentation.

Cependant, un membre du binôme ayant abandonné ses études à l'ESIAL en mars, cet arrangement a été contrecarré. Les tâches à réaliser pour la fin du PIDR se sont alors recentrées sur l'implémentation des leçons.

3 Travail effectué

3.1 Élaboration d'une documentation

Un des premiers travaux entrepris fut la rédaction d'une ébauche de documentation destinée aux développeurs de l'application. La JLM ayant été développée en quasi-totalité par Martin QUINSON et Gérald OSTER, il n'y a pas de documentation prévue pour les utilisateurs extérieurs.

- Comment créer puis ajouter un exercice
- Comment créer une leçon
- Comment créer un monde
- Comment créer un monde
- Une liste des raccourcis créés (cf prochaine section)

3.2 Amélioration de l'interface

La JLM étant un programme complet et entièrement fonctionnel, il ne s'agissait pas ici de la finir ou de la compléter. Toutefois, nous avons pu modifier l'interface de façon à en augmenter la navigabilité par l'ajout de raccourcis clavier et d'un panel affichant l'avancée de l'élève dans sa leçon sur le côté gauche.

Les raccourcis ont été rajoutés de deux manières, toutes deux dépendant de Swing, la bibliothèque graphique utilisée pour la JLM.

La première, la plus simple, utilise a fonction *mnemonic* de Swing. On peut ainsi associer une action d'un composant à un raccourci clavier utilisant la touche ALT. Ainsi, on peut associer le raccourci ALT-R au JButton représentant l'action *Run* dans le programme en lui associant le *mnemonic* approprié. Ces modifications se font donc là ou les composants visés sont déclarés, ici surtout dans la classe *MainFrame.java*

La deuxième, plus subtile, se place à la racine d'un panel et impose le raccourci à tous ses composants enfants (à moins que la combinaison soit déjà réservée). Cette méthode est plus longue mais plus générale car le raccourci n'a pas besoin d'être rattaché à l'action d'un composant particulier. De plus, cette méthode permet d'affecter n'importe quelle combinaison non encore réservée. Ainsi, un appui sur F1 fait apparaitre un panel d'aide.

N.B. Un cas inexpliqué où ce type de raccourci ne fonctionne pas : les CTRL-PageUp ou CTRL-PageDown qui semblent ne pas pouvoir être imposés dans le JEditorPane, soit le composant fils central de notre fenêtre.

L'avancée de l'élève dans la hiérarchie des exercices et des leçons est représentée par un *LessonNavigatorPane*. Auparavant, il s'affichait sous forme de fenêtre lorsque l'utilisateur souhaitait changer d'exercice. Désormais, pour une meilleure navigabilité, il s'affiche à gauche du panel principal.

3.3 Mise en place d'une suite d'exercices

Le but central du projet était d'élaborer une suite d'exercices introduisant progressivement les notions de POO à des débutants. Le travail s'est globalement réparti en trois parties :

- Le choix des exercices, avec une montée progressive en difficulté.
  Nous nous sommes ici très inspirés des travaux des années précédentes.
- L'implémentation des exercices dans la JLM.
- L'élaboration de tests génériques pour vérifier la solution proposée par l'utilisateur.

La description des leçons se fera dans la partie ci-dessous.

4 Leçons de POO

4.1 Descriptif

Dans l'univers BuggleQuest, l'étudiant est confronté au monde familier des Buggles. Face à un problème qu'il doit résoudre avec les indications du sujet, il doit proposer une solution en Java respectant les exigences de la POO. Comme pour l'initiation à la programmation, les effets de la solution proposée sont représentés par l'évolution du monde sur lequel on travaille, ici une grille avec un Buggle, petit animal sympathique.

Alors que les leçons d'initiation à la programmation s'attachaient à compléter un monde déjà existant (avec des algorithmes de parcours de plus en plus compliqués par exemples), le fil conducteur des leçons suivantes est de faire re-créer ce monde par l'étudiant, en commençant par le plus simple, pour arriver à des choses plus compliquées, tout en abordant les notions essentielles de POO.

4.1.1 Création de la classe Buggle - FUDC

Cette première série d'exercices permet à l'élève d'aborder les notions de base de POO en réécrivant la classe Buggle qu'il a déjà appréhendées dans la première leçon de Java. On demande aux élèves d'écrire leurs premières classes, contrairement aux méthodes qu'on leur demandait d'écrire dans l'initiation à la programmation. [1]

Notions abordées
Objet
Classe
Attribut
Constructeur
Setters
Getters

Liste des exercices de cette leçon :

- Classe simple
- Ajout de directions
- Instanciation
- Déplacement

4.1.2 Création des classes Monde et Baggle - FUDC/OOD

Cette seconde série d'exercices permet à l'élève d'approfondir les premiers notions abordées à travers l'écriture de deux autres classes World et Baggle. Elle permet également d'appréhender la collaboration entre les classes. Ainsi, l'élève peut instancier des objets et appeler des méthodes d'autres classes. [2]

Notions abordées
Notions précédentes
Collaboration entre classes
Listes

1. La description de la leçon est suffisamment claire, pour plus de détails, lire *CreateBuggle.html*
2. Idem, voir *CreateBaggle.html* avec un 'a'. :-)

Liste des exercices de cette leçon :

- Création de la classe Baggle
- Création de la classe Monde
- Ajouter un Buggle sur le monde
- Sur un baggle ?
- Ramasser un baggle
- La chasse au baggle
- Ajouter plusieurs baggles sur le monde

4.1.3 Différents types de Buggles

Cette troisième série d'exercice permet à l'étudiant de se familiariser avec les notions d'héritage à travers l'étude de plusieurs types de buggle et l'introduction de nouveaux objets : les murs

Notions abordées
Héritage
Polymorphisme
Interface
Classe abstraite

Liste des exercices de cette leçon :

- Sur un baggle ?
- Polymorphisme
- Des murs abstraits
- Un comportement commun

4.1.4 Sécuriser la vie des buggles

Cette quatrième série d'exercices permet d'aborder plusieurs notions importantes pour écrire un code propre en Java. En particulier la création d'exceptions et d'erreurs, leur lancer et leur capture. [3]

4.2 Aspects techniques

Le principal objectif du PIDR étant d'intégrer de nouvelles leçons à la JLM, nous nous sommes naturellement heurtés à des problèmes techniques. Contrairement à nos habitudes, nous nous retrouvions face à une application déjà intensivement développée par des concepteurs autres que nous. Il nous a fallu être particulièrement consciencieux pour comprendre l'organisation et le fonctionnement du programme, de façon à pouvoir le manipuler et à le modifier.

Notre travail nous a permis d'appréhender deux "hiérarchies" de classes travaillant avec les exercices. Un exercice, ou une leçon [4] comporte de façon visible un sujet (fichier HTML), un monde (ici BuggleQuest), les entités - des "habitants" - de ce monde, la solution proposée par l'élève et la solution effective.

3. Cette série d'exercices à été prévue et rédigée mais pas encore intégrée dans la JLM.
4. La distinction existe (une leçon est un ensemble d'exercices) mais n'est pas importante ici

Pour créer un exercice à partir d'un monde existant, il faut donc créer à la fois une instance d'exercice - contenant le sujet et connaissant le monde - ainsi que l'entité du monde, soit l'élément qu'on va faire interagir avec le monde.[5] Pour cela, nous nous sommes inspirés des leçons déjà écrites et nous avons étendu les classes *jlm.core.model.lesson.Exercise* et *jlm.universe.Entity*.

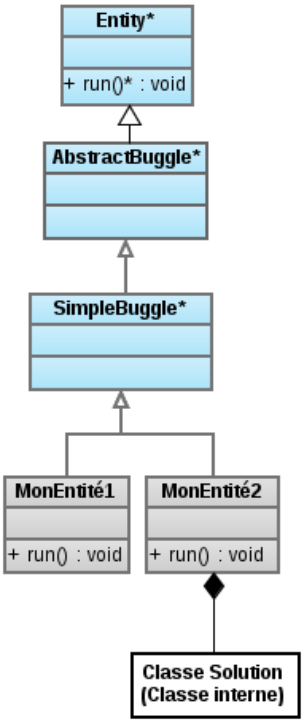


FIGURE 1 – Hiérarchie de l'Entité

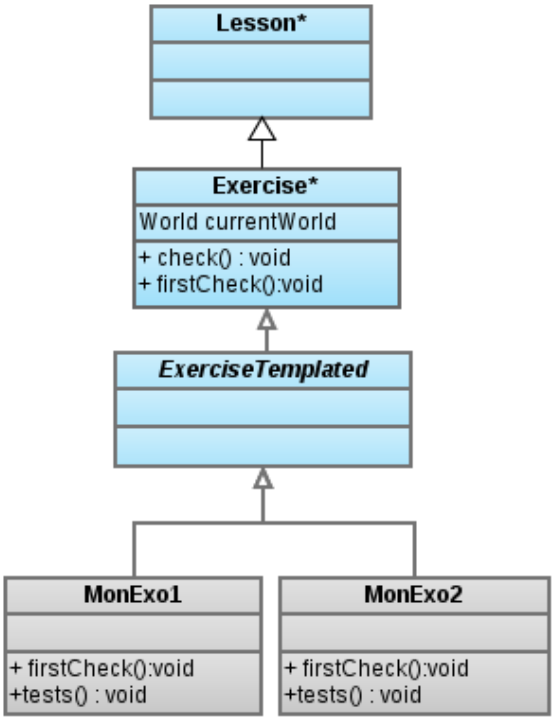


FIGURE 2 – Hiérarchie de l'Exercice

Des tutoriels détaillés pour créer et intégrer exercices et leçons ont été écrits et sont accessibles sur le dépôt Git utilisé : `http://github.com/marionlb/JLM/wiki/How-to`

4.3 Élaboration des tests pour la vérification

Bien-sûr, il ne suffit pas d'écrire une leçon pour pouvoir l'utiliser face à des débutants. Il faut encore vérifier que la solution proposée par l'utilisateur réponde bien au sujet. Pour ce faire, nous pourrions simplement vérifier que le code étudiant corresponde ligne par ligne au code solution proposé par l'enseignant. Cependant, ce système montre très rapidement ses limites : il est tout simplement absurde puisqu'une simple inversion de lignes permettrait de fausser le résultat. Il faut donc trouver une autre solution qui soit fonctionnelle et qui renvoie des messages d'erreurs explicatifs à l'utilisateur lorsque son code ne vérifie pas ce qui est demandé.

Les erreurs de compilation et le résultat de l'exécution sont déjà vérifiées par JLM. Il ne s'agit pas là de re-faire ce qui a déjà été fait, mais d'ajouter une catégorie de tests intermédiaires adaptés aux nouvelles leçons. Puisque ces leçons demandent systématiquement d'écrire des classes, attributs,

5. On parle toujours ici du Buggle puisque ce PIDR utilise le monde BuggleQuest mais on pourrait imaginer des choses tout à fait différentes dans cet univers ou un autre

constructeurs, etc., ces tests seraient chargés de vérifier leur présence et leur adéquation avec la demande. Par exemple, si l'énoncé demande une méthode `int forward()` alors qu'une méthode `void forward()` est demandée, ces tests généreront un message d'erreur approprié.

De plus, ces tests comportent des méthodes automatiques pour tester les getters et les setters [6] simples. Ceux-ci comparent l'état réel des attributs et le retour des méthodes afin de déterminer si l'accès aux attributs a été correctement fait. Ces tests sont appelés cinq fois avec des variables générées au hasard et vérifient donc l'effet de l'appel de ces méthodes. Dans le cas d'une erreur, un message spécifique est généré pour aider l'utilisateur. D'autre part, pour s'adapter au monde torique des Buggles, il est possible d'allouer une tolérance au tests de ces méthodes : ainsi, dans un monde de dimension 7x7, un appel au setter de la position x avec 15 en argument et qui fixera une valeur de 1 (15 modulo 7) sera accepté comme juste pour le test.

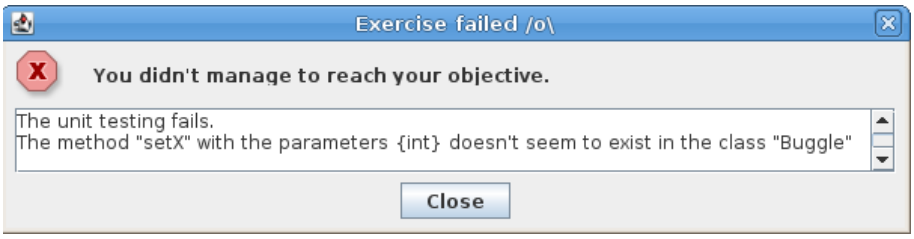
Bien que conçus pour tester les classes créées par les utilisateurs de JLM, ces tests sont génériques et peuvent être appliqués à de nombreuses classes. Cependant, les tests sur les résultats de méthodes (et non leur existence ou leurs types de retour) restent faillibles : ils ne peuvent s'appliquer que dans les cas où les accesseurs sont très simples. Heureusement, c'est très souvent le cas.

Les nouveaux tests peuvent vérifier :

- L'existence d'une classe
- L'existence d'un attribut dans une classe à partir d'un fragment de son nom
- L'adéquation du type d'un attribut avec celui demandé
- L'existence d'une méthode
- L'adéquation des types de retour et de paramètres d'une méthode avec ceux demandés
- L'existence d'un constructeur demandé
- Les retours des accesseurs de la classe (set et get)

Ces tests sont donc utilisés dans chaque exercice des nouvelles leçons. Ils s'exécutent après les tests de compilation, mais avant ceux qui vérifient le résultat d'exécution. Afin de respecter cet ordre, nous avons légèrement modifié le code de la classe abstraite *Exercise* pour lui ajouter une méthode `firstCheck()` vide qui est appelé dans `check()`, la méthode qui vérifie le résultat de l'exécution. Cette méthode est surchargée lorsqu'on veut effectuer d'autres tests avant celui du résultat d'exécution. En cas d'erreur, c'est une *JLMException* qui est levée, et qui est capturée au niveau de `firstCheck()`.
Plusieurs exemples des erreurs retournées par ces vérifications intermédiaires :

FIGURE 3 – Méthode manquante



6. Méthodes d'accès aux attributs de classes. Fondamentales en POO

FIGURE 4 – Constructeur absent

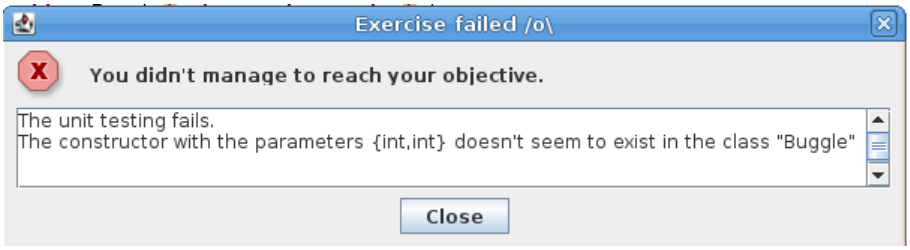


FIGURE 5 – Incohérence de type sur un attribut

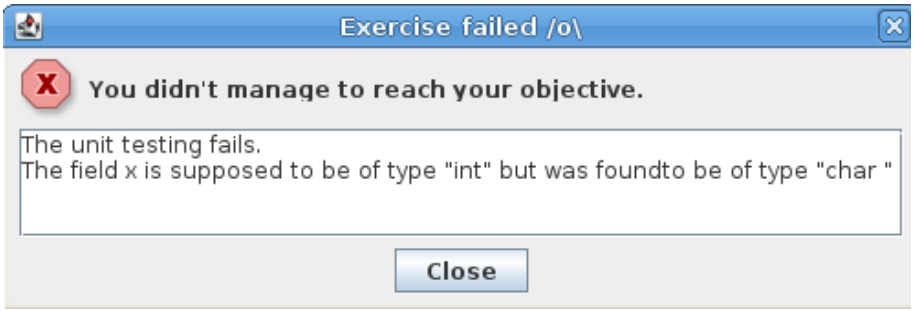
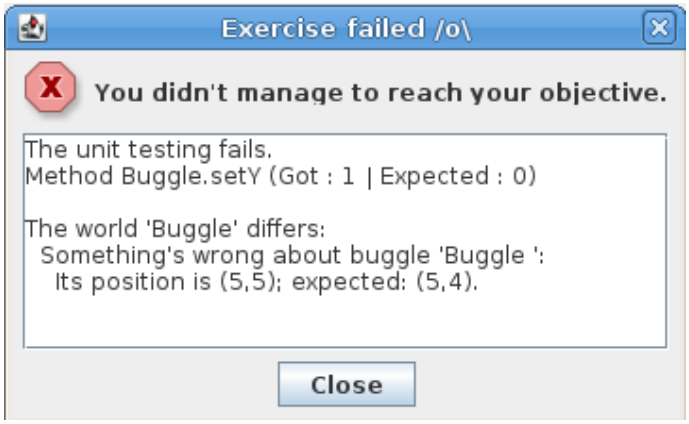


FIGURE 6 – Erreur dans un accesseur



Tous ces tests se trouvent dans la classe *jlm.lessons.oop.Tests.java*. Pour plus d'informations sur leur fonctionnement ou leurs possibilités, se référer à la JavaDoc de cette classe qui est très complète (en anglais).

5 Discussion

5.1 Difficultés rencontrées

La plupart des difficultés - ou plutôt des ralentissements - rencontrés provenaient d'une appréciation trop vague de l'architecture de l'application. En oûtre, nous avons du nous familiariser avec le gestionnaire de version Git. Heureusement, le travail nous ayant déjà été préparé par les PIDRs des années précédentes, nos travaux se sont plus concentrés sur l'implémentation et l'intégration des exercices que leur création - ce qui est une tâche relativement ardue.

5.2 Améliorer la visualisation des résultats

Actuellement, les leçons de POO utilisent la même visualisation que celle des leçons d'initiation à la programmation : le Buggle exécute un code et on compare le monde final au mode solution. Cependant, avec l'approche de la POO, il serait intéressant de voir autre chose.

Par exemple, plutot que de contempler l'évolution de nos entités dans un monde, il pourrait être intéressant de visualiser l'architecture de ce que nous avons créé, ie la liste des attributs de chaque classe, celle des constructeurs et des méthodes. Cette approche pourrait se faire en mode textuel - listes formatées - ou sous forme de diagramme de classes - à l'aide de JPanels.

Cette démarche a déjà été entamée avec un package d'introspection[7] contenant plusieurs classes dont le rôle est de partir d'un objet donné en paramètre et d'en sortir toutes les informations sur sa structure, ie nom de la classe, nombre nom & type des attributs, liens d'héritage, présence de constructeurs, etc. La sortie se fait sous forme de texte mais le back-end est largement exploitable pour construire un affichage graphique sous forme de diagramme de classes.

De même, il pourrait être intéressant de développer une visualisation de l'état de la mémoire avec une représentation des objets sous forme de boîtes, comme ce qui est vu en cours et TD. L'étudiant pourrait ainsi voir s'afficher l'état de la pile et du tas à chaque étape et mieux assimiler les notions vues en cours.

Pour cette approche, il est fortement recommandé de s'inspirer de l'application `animjavaexec`[8] qui gère déjà l'affichage de la pile et le coup-à-coup. Reste à implémenter la façon dont les objets sont représentés dans la pile et le tas (le programme ne gère que les types primitifs). Encore une fois, le code du package jem peut ici être très utile.

5.3 Documentation élaborée

Comme nous avons pu le constater, la documentation est cruciale quand on doit développer une application que l'on a pas créée. Même si nous avons fait l'effort de rédiger quelques tutoriaux pour aider les prochains étudiants à travailler sur la JLM, il reste de nombreux points que nous n'avons abordé ni de près ni de loin. De plus, malgré notre meilleure volonté, il se peut que nos contributions en matière de code ne soient pas aussi limpides que nous l'espérons. C'est pourquoi

7. `http://github.com/marionlb/JLM/tree/master/src/jem`
8. `http://github.com/landrey/animjavaexec`

il nous semble que la documentation est une tâche à ne pas négliger pour les prochains étudiants à travailler sur ce sujet de PIDR.

Références

[1] Loïc Vourc'h & Sébastien Toni, *Rapport de PIDR 2010* ESIAL, Juin 2010.

[2] Sébastien Champagne & Gabriel Larroque, *Rapport de PIDR 2011*, ESIAL, Juin 2011.

Annexe

Glossaire

POO Acronyme pour *Programmation Orientée Objet*. C'est un paradigme de langage de programmation à l'instar des langages impératifs, fonctionnels,...

JLM Acronyme pour *Java Learning Machine*. Plateforme d'apprentissage pour les langages de programmation et en particulier tournée vers le JAVA.

SIGCSE *Special Interest Group on Computer Science Education*. Groupe de réflexion dans le domaine de la pédagogie en informatique.

CS *Control Structures*. Regroupe les compétences en informatique sur l'utilisation des structures de programmation classique comme les boucles et les structures conditionnelles. Première étape de l'apprentissage de la POO.

FUDC *First User Defined Class*. Il s'agit des notions sur la structure et l'utilisation - instanciation de nouveaux objets, invocation des méthodes- d'une classe indépendamment des autres. Deuxième étape de l'apprentissage de la POO.

OOD *Objet Oriented Design*. Partie du paradigme de POO sur l'utilisation des relations possibles entre les classes -Héritage, collaboration,... Troisième étape de l'apprentissage de la POO.

BuggleQuest Un des monde de JLM. C'est une grille où vivent des créatures originales, les buggles. Les buggles se déplacent d'une case à une autre de la grille.

Buggle Entité vivant dans le monde BuggleQuest de JLM.

Swing Bibliothèque graphique utilisée par JLM

Git Gestionnaire de version décentralisé

Table des figures