



DOSSIER INTERPRÉTEUR DU LANGAGE LIR

PROJET PROPOSÉ PAR FRÉDÉRIQUE BARRIOS

Nicolas CAMINADE, Sylvan COURTIOL,
Pierre DEBAS, Heïa DEXTER,
Lucàs VABRE

Sommaire

I	Plan projet	3
II	Spécifications détaillées	4
III	Conception	5
IV	Codage	6
V	Tests	7
	Démarche globale	8
1	Tests du paquetage interpreteurlir.donnees.litteraux	9
1.1	Litteral	9
1.2	Chaine	9
1.3	Entier	9
1.4	Booleen	9
2	Tests du paquetage interpreteurlir.donnees	10
2.1	Identificateur	10
2.2	IdentificateurChaine et IdentificateurEntier	10
2.3	Variable	10
3	Tests du paquetage interpreteurlir.expressions	11
3.1	Expression	11
3.2	ExpressionChaine	11
3.3	ExpressionEntier	11
3.4	ExpressionBoolenne	11
4	Tests du paquetage interpreteurlir	12
4.1	InterpreteurException et ExecuteurException	12
4.2	Contexte	12
4.3	Analyseur	12

5	Tests du paquetage interpreteurlir.programmes	13
5.1	Etiquette	13
5.2	Programme	13
5.3	Les programmes de tests	13
6	Tests du paquetage interpreteurlir.motscles	14
6.1	Commande	14
6.2	EssaiCommande	14
6.3	CommandeCharge	14
6.4	CommandeDebut	15
6.5	CommandeDefs et CommandeFin	15
6.6	CommandeEfface, CommandeLance et CommandeListe	15
6.7	CommandeSauve	15
7	Tests du paquetage interpreteurlir.motscles.instructions	16
7.1	Instruction	16
7.2	InstructionAffiche, InstructionEntre et InstructionSi(Vaen)	16
7.3	InstructionProcedure, InstructionRetour, InstructionStop et InstructionVaen	16
7.4	InstructionVar	16
VI	Conclusion	17
1	Conception et implémentation	18
1.1	Le livrable	18
1.2	Conception	18
2	Organisation du groupe	19
2.1	Travail en binôme	19
2.2	Répartition de la charge de travail	19
2.3	Communication	20
3	Conclusion générale	21
VII	Manuel utilisateur	22

Première partie

Plan projet

Deuxième partie

Spécifications détaillées

Troisième partie

Conception

Quatrième partie

Codage

Cinquième partie

Tests

Démarche globale

Afin de développer l'Interpréteur LIR selon un modèle de cycle de vie itératif, nous avons privilégié la méthode du TDD, Test Driven Development ou développement dirigé par les tests en français.

Ainsi, la majorité des composants de ce logiciel ont été développés selon cette méthode à l'aide des outils de tests écrits lors des TD de Programmation Orientée Objet du semestre 2. Par conséquent, nous n'avons pas utilisé le framework de test JUnit.

Chapitre 1

Tests du paquetage interpreteurlir.donnees.litteraux

1.1 Litteral

Lors des itérations 1, la classe Litteral a été développée comme une classe non-abstrait, en effet nous n'avions pas encore abordé cette notion en cours. Ainsi cette classe a été développée en TDD et nous avons par conséquent effectué les tests unitaires de cette classe.

Cependant à la fin de l'itération 3, dans une optique d'amélioration des codes sources, nous avons passé cette classe en abstrait, ainsi les tests unitaires menés n'avaient plus lieu d'être et ont donc été tout de même conservés en commentaire.

1.2 Chaîne

Les jeux de tests de la classe Chaîne prennent en compte les cas de chaînes vides, la taille maximale des chaînes, leur syntaxe (avec le contenu de la chaîne entre "). Aussi l'opération de concaténation a été testée. Tous les tests menés ont été concluants.

1.3 Entier

La classe Entier est très proche de la classe Integer existant déjà dans le JDK, ainsi son développement a été rapide. À l'instar des tests menés pour la classe Chaîne, tous les tests de la classe Entier notamment des opérations arithmétiques ont été concluants.

Notons, que pour les opérations arithmétiques telles que la division et le reste de la division, le cas particulier de la division par zéro a été testé à part.

1.4 Booleen

La classe Booleen n'a posé aucun problème particulier.

Chapitre 2

Tests du paquetage interpreteurlir.donnees

2.1 Identificateur

La classe Identificateur a été développée en TDD lors de l'itération 1 cependant elle a été passée en abstract lors de l'itération 3, comme pour la classe Litteral, les tests unitaires menés lors de l'itération 1 n'avaient plus lieu d'être et ont été conservés en commentaire. La méthode d'instance compareTo() testée avant le passage de la classe en abstrait et vaut pour les identificateurs d'entier et de chaîne.

2.2 IdentificateurChaine et IdentificateurEntier

Lors des tests unitaires des deux classes, la syntaxe des identificateurs a été testées. Les tests ont été concluants.

2.3 Variable

La classe Variable a été développée lors de l'itération 1 et a donc été testée avec les identificateurs d'entier et de chaîne et seulement avec des valeurs de type Chaine, en effet, la classe Entier ne faisait pas partie de la conception de l'itération 1.

Chapitre 3

Tests du paquetage interpreteurlir.expressions

3.1 Expression

Cette classe Expression a été passée en abstract lors de l'itération 3 cependant les tests des méthodes statiques restent pertinents et ont donc été conservés.

3.2 ExpressionChaine

Lors du développement de la classe ExpressionChaine, l'ambigüité des symboles des opérateurs ("+" et "=") a posé problème. En effet, il fallait déterminer l'emplacement de l'opérateur en ignorant ces symboles s'ils sont contenu dans les constantes littérales. Au fil du développement, les tests ont été concluants.

3.3 ExpressionEntier

Il n'y a rien à signaler sur le développement de ExpressionEntier.

3.4 ExpressionBoolenne

Le développement a été scindé en deux. Un premier groupe a commencer à écrire les interfaces et les tests cependant il a rencontré des difficultés à coder le constructeur, ont été repris par la suite par un autre groupe. Cependant la méthode calculer() n'a pas posé de problème lors du développement.

Chapitre 4

Tests du paquetage interpreteurlir

4.1 InterpreterException et ExecuteurException

Ces deux exceptions sont héritées de RuntimeException et n'ajoute aucun comportement supplémentaire. Par conséquent, leurs tests n'étaient pas déterminants pour la suite du développement de l'interpréteur.

4.2 Contexte

Le développement de la classe Contexte s'est déroulé sans difficultés. Aussi les tests ont été concluants.

4.3 Analyseur

L'Analyseur n'a pas de tests unitaires car tous les tests ont été menés lors de l'intégration. Tests d'intégration effectués en deux parties, la première lors de l'itération 1 avec le test de la prise en charge des commandes et d'instructions exécutées directement, la seconde lors de l'itération 2 pour l'édition de programme.

Chapitre 5

Tests du paquetage interpreteurlir.programmes

5.1 Etiquette

Aucune difficulté n'a été rencontrée lors du développement de la classe Etiquette, aussi les tests ont été concluants.

5.2 Programme

Lors de l'implémentation de la classe Programme, certaines méthodes ne pouvaient être testées directement car il manquait encore des instructions permettant de le faire. Lors de l'implémentation de ces instructions, les tests de celles-ci ont permis de tester également les méthodes de programmes. Ainsi certains tests de Programme n'ont pu être menés que lors de l'intégration avec les instructions ou commandes.

5.3 Les programmes de tests

Lors de l'itération 2, alors que les commandes sauve et charge n'étaient pas encore implémentées, un composant permettait de charger un programme complet au lancement de l'interpréteur pour les démonstrations.

Lors de l'itération 3, après l'implémentation de la commande charge, quatre fichiers contenant un programme écrit en langage LIR ont été écrits pour les démonstrations et tests finaux de l'interpréteur. Il s'agit de l'exemple de programme proposé dans le cahier des charges et des programmes EtatCivil, Median3Entiers et Factorielle.

Chapitre 6

Tests du paquetage interpreteurlir.motscles

6.1 Commande

Le développement de la classe `Commande` a été mené lors de l'itération 1 en TDD, en effet la classe a été passée en classe abstraite lors de l'itération 3. Les tests mené ont été conservés en commentaire.

6.2 EssaiCommande

Lors de l'itération 1, avant l'implémentation de classe `Analyseur`, pour l'intégration des premières commandes `debut`, `defs` et `fin` nous avons utilisé une classe `EssaiCommande`. Une fois la classe `Analyseur` implémentée, l'intégration des autres commandes a été par la suite testée avec.

6.3 CommandeCharge

Le développement de charge n'a pas été simple, en effet, son implémentation a soulevé un problème de conception. La commande charge doit faire appel à l'`Analyseur` cependant celui-ci n'a pas été conçu de façon assez générale pour prendre en compte ce cas de figure. Au vu des délais à tenir, nous avons choisi la solution qui nous paraissait la plus viable. Celle-ci impliquait de recréer des parties de la classe `Analyseur` au sein même de la classe `CommandeCharge`.

Les tests de charge ont encore une particularité, en effet, ils dépendent de la machine sur laquelle les tests sont effectués, il faut donc adapter les tests à la machine utilisée. Cela consiste à avoir sur la machine utilisée des chemins d'accès et des fichiers coïncidant avec ceux des tests.

6.4 CommandeDebut

La commande debut a évolué entre l'itération 1 et 2 avec l'ajout de la remise à zéro du programme en plus de celle du contexte.

6.5 CommandeDefs et CommandeFin

Le développement de ces commandes n'a pas posé de problème, aussi leurs tests ont été concluants.

6.6 CommandeEfface, CommandeLance et CommandeListe

Le développement de ces trois commandes n'a pas posé de problème particulier. Leurs tests ont permis de tester par la même occasion les méthodes de la classe Programme.

6.7 CommandeSauve

Le développement de ces commandes n'a pas posé de problème. À l'instar de charge, la commande sauve nécessite que les chemins pour les tests soient accessibles sur la machine utilisée.

Chapitre 7

Tests du paquetage interpreteurlir.motscles.instructions

7.1 Instruction

Le développement de la classe `Instruction` a été mené lors de l'itération 1 en TDD, en effet la classe a été passée en classe abstraite lors de l'itération 3. Les tests mené ont été conservés en commentaire.

7.2 `InstructionAffiche`, `InstructionEntre` et `InstructionSi(Vaen)`

Le développement de ces instructions n'a pas posé de problème, aussi leurs tests ont été concluants.

7.3 `InstructionProcedure`, `InstructionRetour`, `InstructionStop` et `InstructionVaen`

Le développement de ces quatre instructions n'a pas posé de problème particulier. Leurs tests ont permis de tester par la même occasion les méthodes de la classe `Programme`.

7.4 `InstructionVar`

Le développement de l'instruction `var` a posé un léger problème avec la nécessité que l'expression suivant le mot clé `var` contienne obligatoirement une affectation. Hormis ce souci, le développement de cette instruction n'a pas posé outre problème.

Sixième partie

Conclusion

Chapitre 1

Conception et implémentation

1.1 Le livrable

À l'issue de ce projet, nous avons pu implémenter toutes les fonctionnalités de l'interpréteur LIR telles qu'elles étaient exposées dans le cahier des charges. Notre version de l'interpréteur fonctionne comme attendu par la MOA, bien que la gestion des erreurs et les messages affichés à l'écran auraient gagné à être plus précis et que certaines parties du code mériteraient une optimisation.

1.2 Conception

Ce besoin d'optimisation découle de difficultés rencontrées lors de la conception des classes. Ces difficultés s'expliquent notamment par notre manque d'expérience. Si nous devions refaire ce projet, il est clair que certains des choix que nous avons faits ne seraient pas réitérés.

Commencer directement par générer des diagrammes d'objets en lieu de diagrammes de classes nous aurait certainement permis de gagner quelques heures de travail au moment de la conception initiale. Nous avons cependant fait mieux pour intégrer des notions apprises au cours du projet, comme par exemple le passage en abstraction de certaines superclasses.

Chapitre 2

Organisation du groupe

2.1 Travail en binôme

Lors de chaque itération, nous avons autant que possible privilégié le travail en binôme, en fonction des disponibilités de chacun. Cette modalité nous a permis de nous assurer que tout le monde participait activement au développement et se sentait intégré et valorisé au sein du groupe.

Nous avons aussi fait en sorte de mettre en place une rotation des binômes afin que chaque membre du groupe puisse travailler avec tout le monde. Nous avons ainsi pu nous confronter à d'autres de travailler, partager nos savoirs et nos expériences personnels et assurer une forte cohésion au sein du groupe.

2.2 Répartition de la charge de travail

Malheureusement, le travail en binôme n'est forcément garant d'une répartition efficace de la charge de travail. Cela pose en effet des contraintes cumulatives ; lorsque un membre du groupe a terminé sa tâche, si la suivante nécessite un travail à deux, ce membre devait parfois attendre que son binôme se libère. Il est arrivé qu'un des deux membres d'un binôme prenne du retard sur sa tâche. Cela a occasionnellement posé un frein sur cette modalité de travail.

Devant travailler le weekend, nous nous sommes également heurtés aux aléas des disponibilités personnelles de chacun. Nous avons donc dû composer avec des contraintes familiales, universitaires (devoirs à rendre, révisions,...) ou personnelles. Ces difficultés seraient mitigées dans un contexte professionnel avec des horaires de travail définis dans un contrat.

Nous regrettons aussi de ne pas avoir mis en place un roulement dans les responsabilités (chef de projet, secrétaire, gestionnaire de configuration). Nous avons préféré nous concentrer sur le code.

2.3 Communication

Tout au long du projet, nous avons mis l'accent sur la communication, afin de toujours avoir un aperçu de l'avancée de notre travail. L'utilisation d'un serveur *Discord* dédié au projet a été un outil primordial. En effet, cet outil nous a permis de travailler en binôme en visioconférence, d'organiser des réunions MOE en distanciel.

L'utilisation de « salons » thématiques de conversation a aussi ouvert la possibilité de s'entraider lorsqu'une difficulté se présentait, faire circuler les informations, organiser les réunions, ou plus simplement discuter (moments de convivialité). Grâce à la synchronisation avec le dépôt Github, chaque membre recevait en temps réel les notifications sur l'évolution du projet.

Nous pensons que la communication a été un atout de taille dans la conduite de ce projet. En effet, elle nous a permis de surmonter au mieux les difficultés qui se sont présentées au cours de la conception et du développement de l'interpréteur.

Chapitre 3

Conclusion générale

Nous avons vécu ce projet comme une expérience enrichissante que nous considérons dans l'ensemble comme une réussite. Nous avons pu acquérir et consolider des compétences précieuses au travail d'équipe. Nous tâcherons au cours des prochains projets tutorés de réinvestir nos succès et apprendre de nos échecs.

Septième partie

Manuel utilisateur