UNIVERSITÉ DE ROUEN

Compilateur LLVM

Langage jouet Kawa

Spécification Technique de Besoin 0.1.5

3 décembre 2014

Auteur(s): Kheireddine Berkane, Nasser Adjibi
Relecteur(s): Pierre-Luc BLOT

Version	Date	Changelog
0.1	04/11/2014	Version initiale.
0.1.5	18/11/2014	Définition des cas d'utilisations et des exigences.
0.2	03/12/2014	Modifications par rapport au retour client du 25/11/2014.

Table des matières

1 Objet

LLVM est une infrastructure modulaire permettant la réalisation de chaînes de compilation et conçue pour l'optimisation. Elle met en oeuvre une représentation intermédiaire du code qui permet de découpler les langages de l'architecture. Notre objectif est de réaliser, à l'aide de l'infrastructure LLVM, un compilateur pour un langage jouet, que nous appellerons **Kawa**, ce dernier doit supporter :

- Les classes, les classes abstraites, les interfaces
- L'héritage
- Le polymorphisme
- Le système de types sera composé des types primitifs (int, float, etc.), des classes et des interfaces
- Les instructions de contrôle telles ques (if/else, for, while/do, switch, etc.).
- Les méthodes seront définies de manière identique à Java excepté que les paramètres pourront être préfixés du mot clé VALUE (transmission par valeur au lieu de référence)

1.1 Objectifs techniques

l'objectif à travers le choix du langage **kawa** qui est similaire à **java** (un langage de haut niveau d'abstraction) c'est de permettre une facilité ainsi qu'une rapidité de développement par rapport à d'autres langages de programmation tel que le « C ».Le code Java est compilé dans un langage intermédiaire Bytecode et est exécuté dans un environnement d'exécution JVM (Java Virtual Machine), dans le cadre de notre projet nous allons montrer que l'on peut compiler un code similaire à java (kawa) en code natif sans passer par une machine virtuelle (MV), cette manière de compilation permet d'avoir un code machine en un seul passage à l'encontre de compilateur « javac » qui produit le code binaire en deux passages, un premier passage pour générer de pseudo code (bytecode) et le deuxième passage pour optimiser et traduire localement le bytecode en instructions du processeur de la plate-forme.

Pour réaliser le compilateur du langage **kawa** nous allons utiliser l'infrastructure **LLVM**, à travers cette infrastructure moderne nous allons apprendre la compilation d'un langage évolué dans ces différentes étapes jusqu'à la génération de code machine, car LLVM fournit des librairies facilitant l'écriture de front-end (transformation de code source en une représentation intermédiaire **IR**), ainsi que la partie back-end (transformation de la représentation intermédiaire en code machine).

Nous allons bénéficier dans le cadre de ce projet des différents avantages offerts par **LLVM** et **Clang** (compilateur c++ basé sur llvm) tels que :

- Les messages d'erreurs sont beaucoup plus compréhensibles.
- La rapidité de la compilation ainsi que l'optimisation de la vitesse d'exécution.
- Une architecture en bibliothèque qui est suffisamment modulaire ce qui rend facile le développement et l'intégration de d'autres bibliothèques.
- La portabilité sur d'autres architectures.

2 Documents applicables et de référence

Différents documents de référence :

- Le site LLVM llvm.org.
- Le document de spécification du client Spec1.pdf

3 Terminologie et sigles utilisés

- LLVM : (Low Level Virtual Machine) est une infrastructure de compilateur conçue optimisation à la compilation.
- Kawa : langage jouet qui reprend quelques fonctionnalités de java.
- Clang : compilateur pour le langage c++, son interface de bas niveau se base sur des bibliothèques llvm pour la compilation. Il est utilisé par APPLE.
- GIT : logiciel qui permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus.

— ELF : (Executable and Linkable Format) est un format de fichier binaire utilisé pour l'enregistrement de code compilé (objets, exécutables, bibliothèques de fonctions).
— Ubuntu : Distribution linux sur base Debian.
— C++ : Langage de programmation orienté objet bas niveau.
— POO : Programmation orientée objet.
— Makefile : Fichier regroupant des instructions de compilation avec gestion de dépendances.

4 Exigences fonctionnelles

4.1 Présentation de la mission du produit logiciel

Id	Intitulé	Acteur(s)	Priorité
EF_1	Afficher l'aide	Utilisateur	Indispensable
EF_2	Compiler une application en mode monolithique	Utilisateur	Indispensable
EF_3	Compiler une application en mode partagé	Utilisateur	Secondaire
EF_4	Compiler application utilisant des bibliothèques partagées	Utilisateur	Important
EF_5	Afficher la version du compilateur	Utilisateur	Important
EF_6	Indiquer les chemins des dépendances entre sources et classes déjà compilées	Utilisateur	Important
EF_7	Activer l'affichage en couleur	Utilisateur	Secondaire

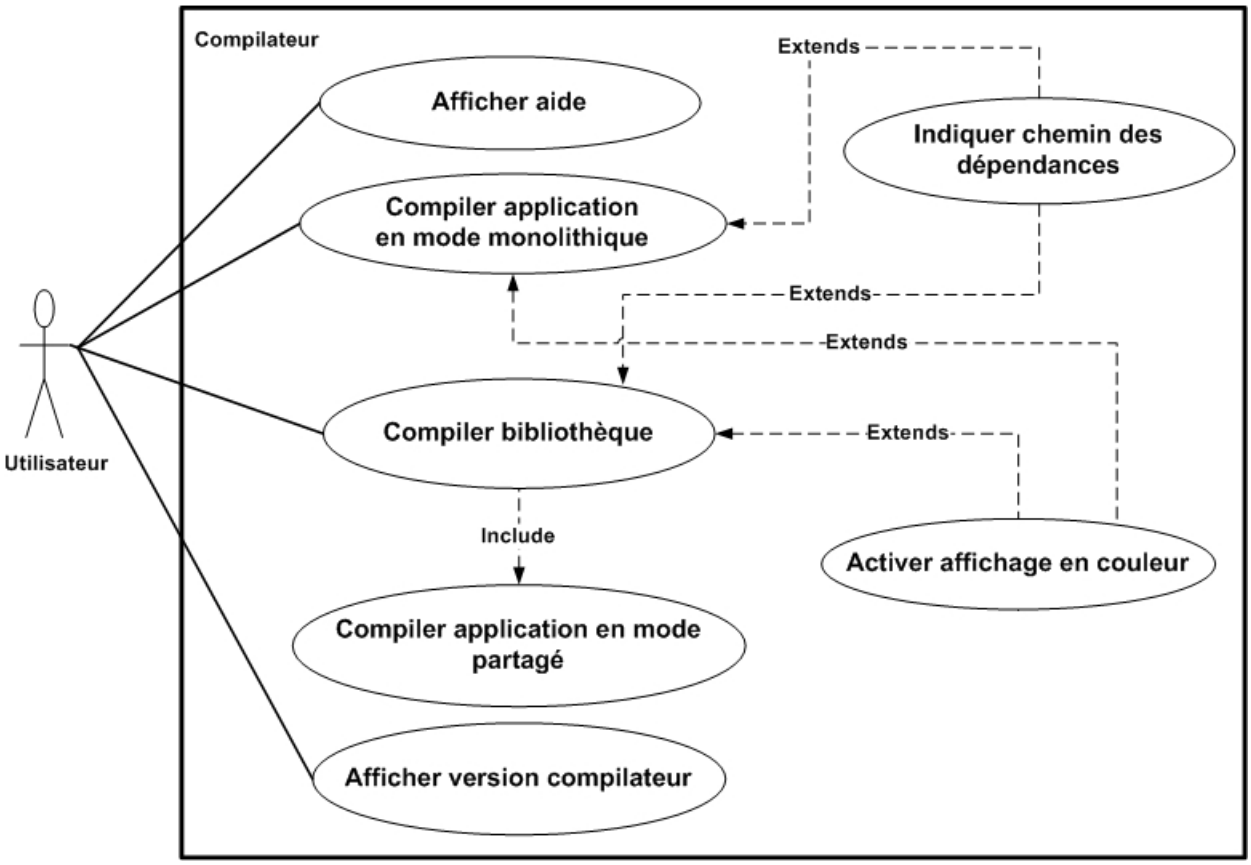

FIGURE 1 – **Cas d'utilisations du compilateur kawa.**

4.2 Cas d'utilisation EF_1

Nom	**Afficher l'aide**
Acteurs concernés	Utilisateur du compilateur
Description	le compilateur affiche la liste des options du compilateur sur la sortie standard à travers une ligne de commande.
Préconditions	L'odre de priorité entre le commutateur de help (-h ou –help) et le commutateur de version (-v ou –version) est définit par la première occurrence de l'un des deux commutateur i-e si nous avons un **-h** avant un **-v**, le compilateur annule tout le reste et affiche le help
Evénements déclenchants	Commutateur de ligne de commande :kawac -h ou –help
Conditions d'arrêt	Action utilisateur permettant de quiter ce mode
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	

4.3 Cas d'utilisation EF_2

Nom	Compiler une application en mode monolithique
Acteurs concernés	Utilisateur du compilateur
Description	L'utilisateur introduit un ensemble de classes kawa afin de les pré-compiler et de générer le tous dans un seul exécutable.
Préconditions	Ensemble de fichiers sources respectant la syntaxe du langage kawa.La non présence des deux commutateurs (-h ou –help) et (-v ou –version) dans la ligne de commande permettant la compilation est obligatoire
Evénements déclenchants	Commutateur de ligne de commande :kawac -m filessources.
Conditions d'arrêt	Fin du programme (succès de la compilation),ou bien un message renvoyé par le compialteur indiquant une erreur rencontrée lors de l'analyse de programme, ainsi que dans certains cas le compilateur ne touve pas les fichiers à compiler.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	Abondant provoqué par l'utilisateur, comme la fermeture du terminal au cours de la compilation.

4.4 Cas d'utilisation EF_3

Nom	Compiler une application en mode partagé
Acteurs concernés	Utilisateur du compilateur
Description	L'utilisateur introduit un ensemble de sources kawa qui peuvent appelés des bibliothèques externes, le compilateur doit être capable de chercher les bibliothèques pour les utiliser ou bien de les recompiler si nécessaire.La distinction entre ce mode et le mode monolithique c'est l'absence du commutateur -m
Préconditions	Ensemble de fichiers source respectant la syntaxe du langage kawa qui peuvent utiliser des bibliothèques partagées.La non présence des deux commutateurs (-h ou –help) et (-v ou –version) dans la ligne de commande permettant la compilation est obligatoire
Evénements déclenchants	Commutateur de ligne de commande :kawac filessources [options]
Conditions d'arrêt	Fin du programme (succès de la compilation),ou bien un message renvoyé par le compialteur indiquant une erreur rencontrée lors de l'analyse de programme, ainsi que dans certains cas le compilateur ne touve pas les fichiers à compiler.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	Abondant provoqué par l'utilisateur,comme la fermeture du terminal au cours de la compilation.

4.5 Cas d'utilisation EF_4

Nom	Compiler application utilisant des bibliothèques partagées
Acteurs concernés	Utilisateur du compilateur
Description	L'utilisateur peut compiler des bibliothèques dynamiques en compilant son application obligatoirement dans un mode partagé,et ceci est possible même si son source n'utilise pas forcément ces bibliothèques externes.
Préconditions	La non présence des deux commutateurs (-h ou –help) et (-v ou –version) dans la ligne de commande permettant la compilation est obligatoire.
Evénements déclenchants	Commutateur de ligne de commande :kawac filessources [options]
Conditions d'arrêt	Fin du programme (succès de la compilation),ou bien un message renvoyé par le compialteur indiquant une erreur rencontrée lors de l'analyse de programme, ainsi que dans certains cas le compilateur ne touve pas les fichiers à compiler.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	Abondant provoqué par l'utilisateur,comme la fermeture du terminal au cours de la compilation.

4.6 Cas d'utilisation EF_5

Nom	Afficher la version du compilateur
Acteurs concernés	Utilisateur du compilateur
Description	L'utilisateur peut savoir la version du compilateur avec le quel compile ses sources et ses bibliothèques à travers un commutateur de ligne de commande.
Préconditions	L'odre de priorité entre le commutateur de help (-h ou –help) et le commutateur de version (-v ou –version) est définit par la première occurrence de l'un des deux commutateur i-e si nous avons un **-v** avant un **-h**, le compilateur annule tout le reste et affiche la version
Evénements déclenchants	Commutateur de ligne de commande :kawac -v ou –version
Conditions d'arrêt	Fin du programme.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	

4.7 Cas d'utilisation EF_6

Nom	Indiquer les chemins des dépendances entre sources et classes déjà compilées
Acteurs concernés	Utilisateur du compilateur
Description	L'utilisateur peut définir les dépendances pour la compilation de son application, en indiquant des chemins entre les sources et des fichiers déjà compilés.
Préconditions	
Evénements déclenchants	Commutateur de ligne de commande :kawac filessources -d path
Conditions d'arrêt	Fin du programme (succès de la compilation),ou bien un message renvoyé par le compialteur indiquant une erreur rencontrée lors de l'analyse de programme, ainsi que dans certains cas le compilateur ne touve pas les fichiers à compiler.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	Abondant provoqué par l'utilisateur, comme la fermeture du terminal au cours de la compilation.

4.8 Cas d'utilisation EF_7

Nom	Activer l'affichage en couleur
Acteurs concernés	Utilisateur du compilateur
Description	L'utilisateur peut activer l'option de l'affichage en couleur, afin de décorer les messages renvoyés par le compilateur dans les différents modes de compilation.
Préconditions	
Evénements déclenchants	Commutateur de ligne de commande :kawac filessource –color
Conditions d'arrêt	Fin du programme.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	

5 Exigences opérationnelles

Id	Intitulé	Priorité

6 Exigences opérationnelles d'interface

Id	Intitulé	Priorité

7 Exigences de qualité

Id	Intitulé	Priorité
EQ_1	les messages d'erreurs renvoyés par le compilateur doivent être explicites, les plus fines possibles et surtout par rapport à ce qu'il s'est passé.	Important

8 Exigences de réalisation

Id	Intitulé et Description	Priorité
EXR_1	**Nommage des fichiers de sortie** : On pourra choisir le nom du fichier à l'issue de la compilation.	Indispensable
EXR_2	**Reconnaissance de la grammaire KAWA** : KAWAC est capable de dire si le code est valide pour la grammaire de KAWA, et émettre des erreurs pour signaler à quelle position dans le texte, et si possible proposer une solution au problème.	Indispensable
EXR_3	**Compilation d'application en monolithique** : KAWAC pourra compiler des fichiers et fournir un exécutable qui n'a pas besoin de bibliothèques externes pour fonctionner. L'ensemble du code donné en entrée devra fournir une méthode main, qui sera le point d'entrée de l'application.	Indispensable
EXR_4	**Compilation d'application partagée** : KAWAC pourra compiler des fichiers et fournir un exécutable. L'exécutable s'il le faut dépendra de ressources externes pour pouvoir fonctionner. Le code en entrée devra fournir une méthode main, qui sera le point d'entrée de l'application.	Important
EXR_5	**Compilation de bibliothèque partagée** : KAWAC pourra compiler des fichiers et fournir une nouvelle bibliothèque. La bibliothèque s'il le faut dépendra de ressources externes pour pouvoir fonctionner. Le code en entrée devra ne pas contenir de methode main.	Important
EXR_6	**Gestion des mots clés** : KAWA spécifie dans sa grammaire une liste de mots réservés. Ces mots sont utilisés par le développeur pour des actions prédéfinies, et ne peuvent être utilisés comme nom de méthodes, d'attributs ou de variables.	Indispensable
EXR_7	**Fichier de sortie au format ELF** : Le compilateur utilisera le format ELF pour la production des fichiers en sortie. Les fichiers contiendront une section contenant les information permettant la résolution des liens d'appel de fonctions. Les fichiers exécutables utiliseront ces informations pour monter en mémoire les références des bibliothèques externes lors de la phase d'édition des liens.	Indispensable
EXR_8	**Gestion de la mémoire** : La gestion de la mémoire est automatisée. Les réservations et libérations des espaces mémoires utilisés par les objets s'effectuent sans une intervention directe du développeur. Les objets non référencés sont susceptibles d'être désalloués. Une demande d'allocation mémoire ne peut être est faite par l'utilisateur que grâce au mot clé **new** permettant d'instancier un objet.	Important
EXR_9	**Importation de packages** : Le développeur peut utiliser des entités présents dans des packages externes, grâce au mot clé **import**.	Important
EXR_10	**Déclaration de package** : Le développeur déclarer un package grâce au mot clé **package**.	Indispensable

EXR_11	**Reconnaissance et compilation de classes** : KAWAC est capable de reconnaître et compiler des classes écrites en KAWA. La déclaration d'une classe se fait grâce au mot clé **class**. Une classe est déclarée dans un fichier **.kawa**, et rend en sortie un fichier **.klass**. Une classe permet de définir des attributs, des constructeurs et des méthodes. Elle peut être instanciée et utilisée par une ou plusieurs applications. Une classe peut dériver une classe abstraite ou une autre classe et implémenter plusieurs interfaces(Veuillez consulter la grammaire KAWA en annexe pour la syntaxe).	Indispensable
EXR_12	**Reconnaissance et compilation de classes abstraites** : KAWAC est capable de reconnaître et compiler des classes abstraites écrites en KAWA.La déclaration d'une classe abstraite se fait grâce aux mots clés abstract class. Une classe abstraite est déclarée dans un fichier **.kawa**, et rend en sortie un fichier **.klass**. Contrairement à une classe normale, une classe abstraite ne peut être instanciée. Elle est faite pour être dériver par une autre classe. Cependant, en plus des fonctionnalités d'une classe normale, une classe abstraite peut définir des prototypes de méthodes qui devront être implémentées par les classes filles de la classe abstraite.(Veuillez consulter la grammaire KAWA en annexe pour la syntaxe).	Indispensable
EXR_13	**Reconnaissance et compilation d'interfaces** : KAWAC est capable de reconnaître et compiler des interfaces écrites en KAWA. La déclaration d'une interface se fait grâce au mot clé **interface**. Une interface est déclarée dans un fichier **.kawa**, et rends en sortie un fichier **.klass**. Une interface ne peut que déclarer les prototypes des méthodes que devront implémenter les classes qui implémenteront l'interface. Ces dernières ne peuvent être que de portée publique et sont abstraites. Une interface ne contient ni attribut, ni constructeur. Une interface est destinée à être implémentée par une classe ou partiellement par une classe abstraite. Le développeur utilisera le mot clé **extends** pour spécifier l'implémentation d'une ou plusieurs interfaces(Veuillez consulter la grammaire KAWA en annexe pour la syntaxe).	Indispensable
EXR_14	**Prise en charge du polymorphisme ad-hoc** : Si plusieurs méthodes portent le même nom, KAWA est capable de déterminer la méthode à appeler lors de l'exécution de l'application en fonction de la signature de la méthode.	Important
EXR_15	**Prise en charge du polymorphisme de sous-type** : KAWA est capable de déterminer lors de l'exécution la méthode en fonction du type dynamique de la classe appelante. KAWAC n'autorise pas le cast d'objets.	Important
EXR_16	**Dérivation d'entités** : En utilisant le mot clé **extends** dans la déclaration d'une classe, d'une classe abstraite ou d'une interface, l'utilisateur est capable de dériver ou d'implémenter une classe ou des interfaces. Une classe peut dériver une autre classe ou une classe abstraite. Si non abstraite dérive une classe abstraite ou une interface, elle doit implémenter toutes les méthodes abstraites de la classe qu'elle dérive. Une classe ne peut dériver qu'une classe à la fois, mais peut implémenter plusieurs interfaces. Une interface peut dériver plusieurs interfaces, mais ne peut dériver une classe.	Important

EXR_17	**Mécanisme de constructeur** : Chaque classe, pour être instancié, fournit une ou plusieurs méthodes qui permettront son instanciation. KAWAC fournira un constructeur si aucun constructeur n'est pas définit. L'instanciation se fait grâce au mot clé **new**, et retourne une référence vers un espace mémoire stockant l'objet. Une classe abstraite peut définir un constructeur, mais ne peut instancier. Une interface n'a pas de constructeur.(Veuillez consulter la grammaire KAWA en annexe pour la syntaxe).	Indispensable
EXR_18	**Mécanisme de finalisation** : Chaque objet fournit une méthode qui sera appelée lors de sa destruction par le garbage collector. Si aucune méthode n'est définie par le développeur, KAWAC en fournira une par défaut.	Secondaire
EXR_19	**Reconnaissance et définition de méthode et de variables** : On peut déclarer un bloc d'instructions paramétrable s'exécutant s'il est appelé.On peut définir des variables temporaires et dont la portée sera limitée au bloc dans lequel elles ont été déclarées. Les variables locales ne sont pas des attributs et ne sont plus accessibles à la fin du bloc d'instruction les déclarant.	Indispensable
EXR_20	**Reconnaissance et définition d'attribut d'objet** : On peut déclarer des champs propres à chaque objet. Les espaces attribués à chaque attribut ne sont accessibles que durant la période de vie de l'objet, par les membres de l'objet ou par un programme externe si l'attribut est déclaré public.	Indispensable
EXR_21	**Définition d'attributs statiques** : On peut définir des espaces mémoires associés aux classes. Les objets instanciant ou dérivant la classe où a été déclaré l'attribut et si la visibilité le permet, pointeront vers la même adresse pour cet attribut. Une méthode statique ne peut accéder aux attributs ou méthodes qui ne sont pas statiques.	Important
EXR_22	**Définition d'attribut de constantes** :Un attribut constant ne peut être modifié après affectation. Il doit avoir été initialisé d'être utilisable par une autre opération que l'affectation.	Secondaire
EXR_23	**Définition d'attributs, de variables ou de méthodes de type value** : En définissant un attribut avec le mot clé **value**, on aura accès non pas a une référence vers un espace mémoire stockant les données, mais directement a un bloc de données. Dans le cas d'une méthode, la méthode renvoie un bloc de données représentant le résultat.	Important
EXR_24	**Définition méthode a référence** : La méthode renvoie une référence vers un espace mémoire contenant les données de l'objet renvoyé.	Indispensable
EXR_25	**Définition méthode a sans référence** : La méthode ne retourne rien.	Indispensable
EXR_26	**Définition méthode finale** : La méthode ne peut être surchargée ou redéfinie.	Secondaire
EXR_27	**Définition méthode statique** : La méthode peut être accessible à partir du nom de la classe. Une méthode statique ne peut faire appel aux attributs non statiques de sa classe.	Important
EXR_28	**Héritage de méthode** : Une classe dérivant une autre copiera toute les méthodes déjà définit dans l'arborescence de ses ancêtres. Mais ne pourra y accéder que si la visibilité le permet.	Indispensable

EXR_29	**Héritage d'attribut** : Une classe dérivant une autre copiera tout les attributs déjà définit dans l'arborescence de ses ancêtres si la visibilité le permet. On ne peut pas redéfinir un attribut déjà définit dans la classe ou interface dérivée.	Indispensable
EXR_30	**Redéfinition de méthode** : Une méthode peut être définie avec le nom d'une autre existence, en renvoyant le même type de valeur, ainsi que les mêmes paramétrés dans une autre de l'arborescence de la classe ayant premièrement définit.	Important
EXR_31	**Surcharge des méthode** : Une méthode peut être définie avec le nom d'une autre existence, en renvoyant le même type de valeur, mais avec des paramètres différents.	Important
EXR_32	**Concepte de visibilité :** A l'aide des mots clés, on peut définir la visibilité des attributs et méthodes des classes sur plusieurs niveaux. Privée, pour empêcher l'accès en dehors de la classe. Protégée, pour empêcher l'accès en dehors de la classe, sauf pour les hérités de la classe ou de l'interface. Publique pour autoriser l'accès à tous.	Important
EXR_33	**Portée de variable** : Une variable n'est effective qu'à l'intérieur du bloc à dans lequel elle a été déclarée.	Indispensable
EXR_34	**Gestion des exceptions** : On est capable de provoquer volontairement ou non, une execption qui se propagera dans le programme, jusqu'a ce qu'elle soit rattrapée et traitée par le programme. Si une exception n'est pas traité, l'application devra s'arreter et afficher un message d'erreur sur la sortie des erreurs.	Secondaire
EXR_35	**Reconnaissance des expression conditionnelles** : KAWA admet des expressions conditionnelles. Le programme exécute un bloc d'instruction donné ou un autre selon des condition définies par le développeur.	Imporant
EXR_36	**Reconnaissance des expression de bouclages** : KAWA admet des expressions de bouclage. Le programme répétera l'exécution d'un bloc d'instruction donné tant qu'une condition définies par le développeur est correcte .	Important
EXR_37	**Bloc de classe** : La définition du corps de l'objet se fera a l'intérieur d'un bloc délimité par { et }. Le bloc de classe est le bloc générale contenant toutes les déclarations liées à une classe. A part les importations de packages, la déclaration de l'entête de la classe ou interface ou les commentaires, aucun autre élément ne doit être présent dans le fichier. On ne peut définir qu'une classe par fichier et KAWA n'autorise pas la déclaration de classe imbriquée.	Indispensable
EXR_38	**Bloc d'instruction** : On peut définir une suite d'instructions délimité par { et }. Un bloc d'instruction peut contenir d'autres blocs.	Indispensable
EXR_39	**Reconnaissance des commentaires** : Le développeur peut laisser des commentaires dans le code KAWA. Ces commentaire seront reconnus et ignorés par KAWAC lors de la compilation.	Important