UNIVERSITÉ DE ROUEN

Compilateur LLVM

Langage jouet Kawa

Spécification Technique de Besoin 0.4

18 janvier 2015

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

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.
0.3	10/12/2014	— Les parties événements déclenchants ont été détaillées. — Les parties flots d'exceptions, ainsi que les conditions d'arrêts de toutes les exigences fonctionnelles ont été détaillées. — L'ajout des exigences opérationnelles d'interface. — Correction des erreurs signalées lors de la réunion client 04/12/2014.
0.4	22/12/2014	— Les parties événements déclenchants ont été modifiées et structurées sous forme d'une liste. — Les parties flots d'exceptions, ainsi que les conditions d'arrêts de toutes les exigences fonctionnelles ont été été modifiées et structurées sous forme d'une liste. — La suppression des exigences opérationnelles d'interface après une remarque faite par le prof du TP gestion de projet. — Changement de priorité de l'exigence fonctionnelle EF_4 en secondaire car elle dépend de l'exigence EF_3 qui est secondaire. — Correction des erreurs signalées lors du retour client par mail le 19/12/2014.
0.5	18/01/2015	— Élimination des exigences de réalisations qui correspondent aux exigences fonctionnelles afin d'éviter les redondances inutiles. — La partie des exigences fonctionnelles a été détaillée ainsi que les scénarios des exceptions ont été spécifiés afin de faciliter les tests après.

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 bibliothèques 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 en matière de détails ainsi que de précision par rapport à d'autres messages d'erreurs renvoyés par un autre compilateur, exemple (GCC).
- La rapidité en temps de la compilation ainsi que l'optimisation de la vitesse d'exécution, clang est trois fois plus rapide que GCC, de plus il consomme moins de ressources (faible utilisation de la mémoire).
- Une architecture en bibliothèque du compilateur qui est suffisamment modulaire ce qui rend facile le développement et l'intégration d'autres bibliothèques au sein du compilateur.
- 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 pour l'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 une bibliothèques partagée	Utilisateur	Secondaire
EF_5	Afficher la version du compilateur	Utilisateur	Important
EF_6	Indiquer les chemins des dépendances entre le source de l'application et des modules (classe/interface) externes	Utilisateur	Secondaire
EF_7	Activer l'affichage en couleur	Utilisateur	Secondaire

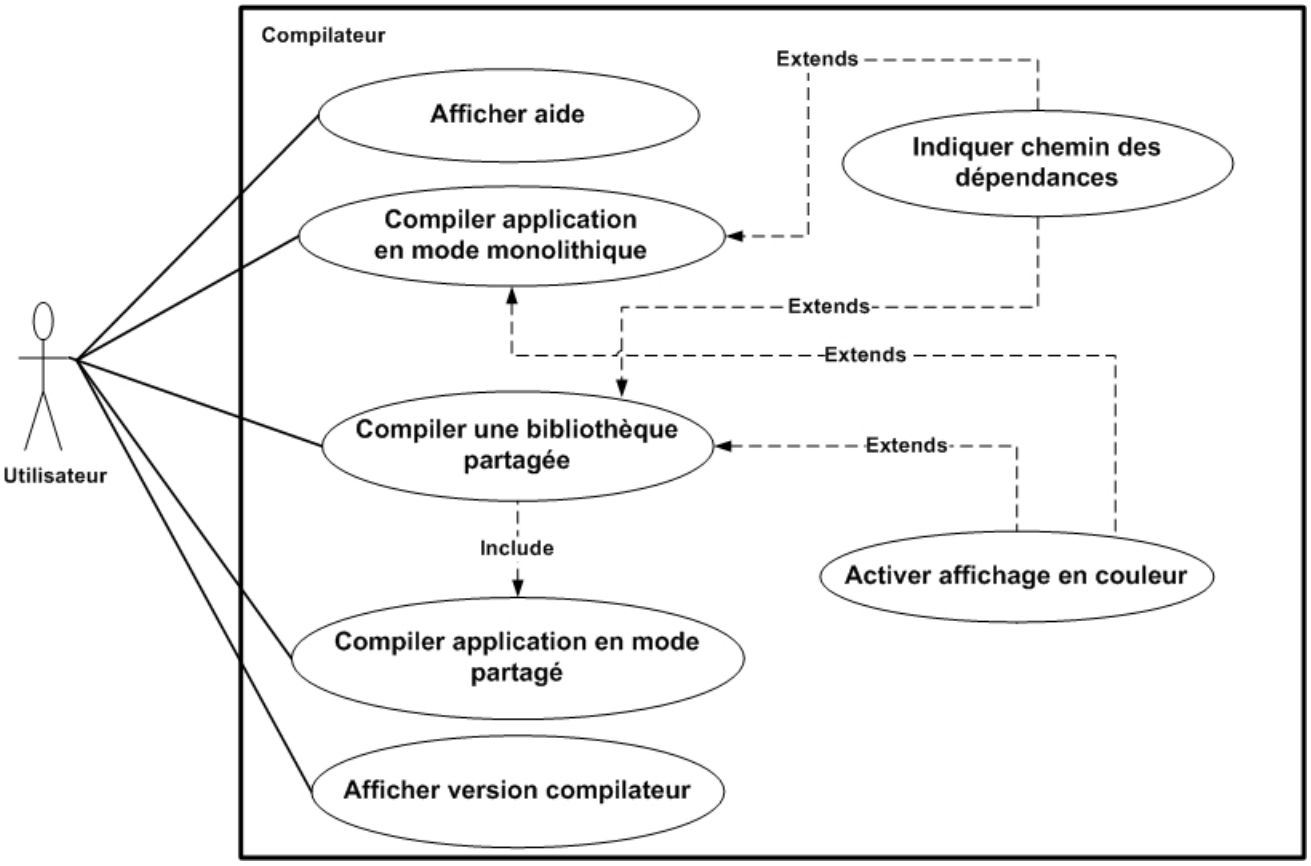

FIGURE 1 – **Cas d'utilisations du compilateur kawa.**

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'ordre de priorité entre le commutateur de help (-h ou --help) et le commutateur de version (-v ou --version) est définit par le commutateur en premier paramètre. Le commutateur -h/--help est le premier paramètre obligatoirement.
Evénements déclenchants	Commutateur de ligne de commande :kawac -h ou --help
Conditions d'arrêt	L'aide a été affichée correctement et la main est redonnée à l'utilisateur pour entrer d'autres commutateurs.
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 modules (classe/interface) kawa afin de les précompiler et de générer une application sous forme d'un seul exécutable.
Préconditions	Afin de permettre la compilation dans ce mode monolithique il faut : — Indiquer l'emplacement des fichiers sources de l'application. — Introduire le commutateur **-m** en premier paramètre dans la ligne de commande permettant la compilation. — L'absence des deux commutateurs (-h ou --help) et (-v ou --version) en premier paramètre.
Evénements déclenchants	Afin de pouvoir lancer la compilation dans ce mode l'utilisateur doit : — Indiquer au moins un module (une classe avec une méthode main qui constitue le point d'entrée d'une application). — Introduire une ligne de commande avec un commutateur d'action : kawac -m filessources.
Conditions d'arrêt	— La compilation s'est terminée sans erreurs dans ces différentes étapes d'analyses. — La production d'un unique fichier exécutable sous le format d'ELF qui ne dépend d'aucune bibliothèque kawa ainsi que le nom du fichier exécutable correspond au nom du module contenant la méthode main.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	La compilation peut être interrompue pour divers raisons, on cite : — **EF_2_EXC_1** : Le fichier de sortie n'a pu être créé. — **EF_2_EXC_2** : Le code source d'un des modules de l'application comporte une erreur syntaxique, l'erreur rencontrée pendant cette étape d'analyse est renvoyée sur la sortie standard du compilateur. — **EF_2_EXC_3** : Le code source d'un des modules (classe/interface) dont dépend le programme est introuvable. Un message renvoyé par le compilateur indique le nom du (ou des) module(s) manquants sur la sortie standard. — **EF_2_EXC_4** : Le code source d'un des modules de l'application comporte une erreur sémantique, exemple : l'incompatibilité de type statique lors d'une opération d'affectation. — **EF_2_EXC_5** : Aucun des modules indiqués ne comporte le point d'entrée (main). — **EF_2_EXC_6** : Plusieurs méthodes main ont été trouvées parmi les classes indiquées dans le source, le compilateur renvoie la liste des points d'entrée trouvés sur la sortie standard.

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 appeler des bibliothèques externes. Le compilateur doit être capable de trouver les bibliothèques pour les utiliser ou bien de les recompiler si nécessaire. La différence entre ce mode et le mode monolithique est que le fichier exécutable et la bibliothèque sont indépendants avant le lancement de l'application, et peuvent être maintenus séparément à l'encontre du mode monolithique qui génère un seul exécutable regroupant le tout. Pour les bibliothèques partagées l'édition des liens se fait à l'exécution (au moment du lancement du programme) au contraire des bibliothèques statiques où l'édition des liens se fait dans la phase de compilation.
Préconditions	Afin de permettre la compilation dans ce mode il faut : — Spécifier l'emplacement des fichiers sources de l'application. — L'absence du commutateur **-m** en premier paramètre. — L'absence des deux commutateurs (-h ou --help) et (-v ou --version) en premier paramètre.
Evénements déclenchants	Afin de pouvoir lancer la compilation dans ce mode l'utilisateur doit : — Indiquer au moins un module (une classe avec une méthode main qui constitue le point d'entrée d'une application). — Introduire une ligne de commande sans commutateur d'action (fonctionnement par défaut) : kawac filessources [options]. L'utilisateur peut éventuellement compiler des fichiers des modules qui ne sont pas forcément en relation avec l'application à compiler.
Conditions d'arrêt	— La compilation s'est terminée sans erreurs dans ces différentes étapes d'analyses. — La production d'un unique fichier exécutable sous le format d'ELF et qui n'est pas sous l'extension (.so), le nom du fichier exécutable correspond au nom du module contenant la méthode main.

Description du flot d'événements principal:

Acteur(s)	Système

Flots d'exceptions:	La compilation peut être interrompue pour divers raisons, on cite : — **EF_3_EXC_1** : Le fichier de sortie n'a pu être créé. — **EF_3_EXC_2** : Le code source d'un des modules de l'application comporte une erreur syntaxique, l'erreur rencontrée pendant cette étape d'analyse est renvoyée sur la sortie standard du compilateur. — **EF_3_EXC_3** : Un module (classe ou interface) de l'application fait référence à un symbole externe introuvable, soit le paquetage externe contenant le symbole n'existe pas, ou bien il n'a pu être compiler à cause d'une erreur. — **EF_3_EXC_4** : Le code source d'un des modules de l'application comporte une erreur sémantique, exemple : l'incompatibilité de type statique lors d'une opération d'affectation. — **EF_3_EXC_5** : Aucun des modules indiqués ne comporte le point d'entrée (main).

4.5 Cas d'utilisation EF_4

Nom	Compiler une bibliothèques partagée
Acteurs concernés	Utilisateur du compilateur
Description	L'utilisateur peut compiler des bibliothèques partagée en introduisant le nom du paquetage contenant les sources de la bibliothèque, la bibliothèque elle même peut contenir des sous bibliothèques (sous paquetages), pour notre cas d'étude chaque sous paquetages aura son propre fichier (.so). Le compilateur produit du code compilé destiné à être partagé entre plusieurs différents programmes, l'extension de la bibliothèque produite sera (.so => Shared Object) qui est un unique fichier exécutable pour tout le paquetage. On ne peut compiler une bibliothèque partagée que dans un mode partagé avec la non présence cette fois ci de la méthode main.
Préconditions	Afin de permettre la compilation d'une bibliothèque partagée expicitement il faut : — Indiquer le nom du répertoire contenant les fichiers sources de la bibliothèque. — L'absence du commutateur **-m** en premier paramètre, pour indiquer qu'on est dans un mode paratgé. — L'absence des deux commutateurs (-h ou --help) et (-v ou --version) en premier paramètre.
Evénements déclenchants	Afin de pouvoir lancer la compilation de la bibliothèque l'utilisateur doit : — Indiquer le nom du paquetage contenant les sources de la bibliothèque. — Introduire la ligne de commande sans les commutateurs d'actions : kawac filessources [options].
Conditions d'arrêt	La compilation s'est terminée sans erreurs dans ces différentes étapes d'analyses et a comme résultats : — La production d'une bibliothèque dynamique dont le nom et l'emplacement correspondent à ceux du paquetage indiqué par l'utilisateur. — La bibliothèque va contenir les symboles de liaison pour tous les éléments publiques du paquetage.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	La compilation peut être interrompue pour divers raisons, on cite : — **EF_4_EXC_1** : Le répertoire du paquetage est vide. — **EF_4_EXC_2** : Erreur rencontrée lors de l'analyse syntaxique de l'un des modules du paquetage. — **EF_4_EXC_3** : Erreur rencontrée lors de l'analyse sémantique de l'un des modules du paquetage. — **EF_4_EXC_4** : Un module (classe ou interface) de la bibliothèque fait référence à un symbole externe introuvable, soit le paquetage externe contenant le symbole n'existe pas, ou bien il n'a pu être compiler à cause d'une erreur. — **EF_4_EXC_5** : Un point d'entrée (main) a été trouvé dans l'un des modules du paquetage.

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	La version est affichée sur la sortie standard du compilateur.
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**
Acteurs concernés	Utilisateur du compilateur
Description	Afin de permettre la définition des schémas de dépendances entre le source de l'application et des modules externes (peuvent être des bibliothèques déjà compilées) il faut : — Spécifier le chemin vers le(s) module(s) externe(s). — L'absence du commutateur **-m** en premier paramètre, car on est obligatoirement dans un mode de compilation en partagé — L'absence des deux commutateurs (-h ou --help) et (-v ou --version) en premier paramètre.
Préconditions	
Evénements déclenchants	Afin de pouvoir lancer la compilation avec les différentes dépendances en interne : — L'utilisateur introduit des schémas de dépendances grâce au commutateur : kawac filessources -d path. — L'utilisateur indique les sources de son appelication.
Conditions d'arrêt	Fin de la compilation sans erreurs dans ces différentes étapes d'analyses, ainsi que la reconnaissance de tous les schémas de dépendances introduit par l'utilisateur.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	La compilation peut être interrompue pour divers raisons, on cite : — **EF_6_EXC_1** : Le module indiqué dans le chemin de dépendance est introuvable. — **EF_6_EXC_2** : Erreur rencontrée lors de l'analyse syntaxique de l'un des modules du code source. — **EF_6_EXC_3** : Erreur rencontrée lors de l'analyse sémantique de l'un des modules du du code source. — **EF_6_EXC_4** : Le source contenant le point d'entrée est introuvable, un message d'erreur est renvoyé par le compilateur sur la sortie standard.

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 la sortie standard dans les différents modes de compilation.
Préconditions	
Evénements déclenchants	Commutateur de ligne de commande :kawac filessource --color
Conditions d'arrêt	Le code présent sur la sortie standard a été colorié.
Description du flot d'événements principal:	
Acteur(s)	**Système**
Flots d'exceptions:	

5 Exigences opérationnelles

Pas de spécifications concernant cette section.

Id	Intitulé	Priorité

6 Exigences opérationnelles d'interface

Pas de spécifications concernant cette section.

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	**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_4	**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. Les fichiers compilés se placeront sur le dique de stockage selon le chemin définit par leur paquet et nom de classe. Un élément ne définissant pas son paquet sera compilé dans le dossier dans lequel il se trouve.	Indispensable
EXR_5	**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_6	**Résolution de nom de classe** : le mot clé import permet à l'utilisateur de spécifier le chemin d'accès à une classe. L'utilisateur n'aura plus à specifier le paquet la classe à chaque utilisation et pourra se limiter au nom de la classe. Si le dévellopeur se sert de classes de differents paquets mais de noms identiques, il sera obliger de spécifier à chaque fois le chemin d'accès de la classe. Si une ambiguïté est detectée, la compilation échouera et affichera une message d'érreur.	Important

EXR_7	**Déclaration de paquets** : le développeur pourra spécifier le chemin du paquet auquel appartient la classe qu'il est entrain de définir grâce au mot clé **package**. La classe devra se trouvrer dans une arborescence de dossiers en accord avec le paquet	Indispensable
EXR_8	**Espace de nommage de classes** : l'emploie du mot clé **package** permet de spécifier un espace de nommmage pour les classes et interfaces. L'utilisateur pourra nommer les classes de son paquet avec le même nom que des classes de à l'exterieur de du paquet. Cependant deux classes du même pacquet ne peuvent pas porter le même nom.	Indispensable
EXR_9	**Reconnaissance et compilation de classes concrètes** : KAWAC est capable de reconnaître et compiler des classes écrites en KAWA. La déclaration d'une classe concrète se fait grâce au mot clé **class**. Une classe concrète est déclarée dans un fichier **.kawa**, et rend en sortie un fichier **.klass** de même nom que la classe. 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_10	**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** de même nom que la classe, 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_11	**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** de même nom que l'interface. 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_12	**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_13	**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_14	**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_15	**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_16	**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_17	**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. Une varible ne peut pas avoir le même nom q'une méthode ou un attribut.	Indispensable
EXR_18	**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_19	**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_20	**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_21	**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_22	**Définition méthode a référence** : la méthode renvoie une référence vers un espace mémoire contenant les données de l'objet renvoyé.	Indispensable
EXR_23	**Définition méthode a sans référence** : la méthode ne retourne rien.	Indispensable
EXR_24	**Définition méthode finale** : la méthode ne peut être surchargée ou redéfinie.	Secondaire

EXR_25	**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_26	**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_27	**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 peut redéfinir un attribut déjà définit dans la classe ou interface dérivée.	Indispensable
EXR_28	**Redéfinition de méthode** : une méthode peut être définie avec le nom, renvoyant le même type de valeur et les mêmes paramétres qu'une autre méthode de l'une des classe de l'arborescence dont la classe actuelle dérive. Une méthode ne peut pas redéfinir une méthode creer de la classe dont elle est issue.	Important
EXR_29	**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_30	**Concepte de visibilité** : à l'aide des mots clés, on peut définir la visibilité des attributs et méthodes des classes sur plusieurs niveaux. Privée avec le mot clé **private**, pour empêcher l'accès du champ ou de la methode en dehors de la classe. Dans le cas d'une définition de classe, on empeche l'utilisation par des éléments n'appartenant pas au même paquage. Publique avec le mot clé **public** autorise l'accès par tous. Une déclaration de classe, interface, méthode, constructeur ou attribut doit specifier une visibilité. La compilation enverra une erreur sinon.	Important
EXR_31	**Portée de variable et résolution de noms de variable** : une variable n'est effective qu'à l'intérieur du bloc à dans lequel elle a été déclarée. L'appel d'une variable dans un bloc remontera l'arborescence des blocs d'instructions et choisira l'occurrence la plus proche. Si aucune occurence n'est trouvée le compilateur émèttra une erreur.	Indispensable
EXR_32	**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_33	**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_34	**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_35	**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 la spécification du chemin paquet et les instructions de résolution des noms de classes, 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_36	**Bloc d'instruction** : on peut définir une suite d'instructions délimité par { et }. Un bloc d'instruction peut contenir d'autres blocs.	Indispensable
EXR_37	**Reconnaissance des commentaires** : le développeur peut laisser des commentaires dans le code KAWA. Ces commentaires seront reconnus et ignorés par KAWAC lors de la compilation.	Important