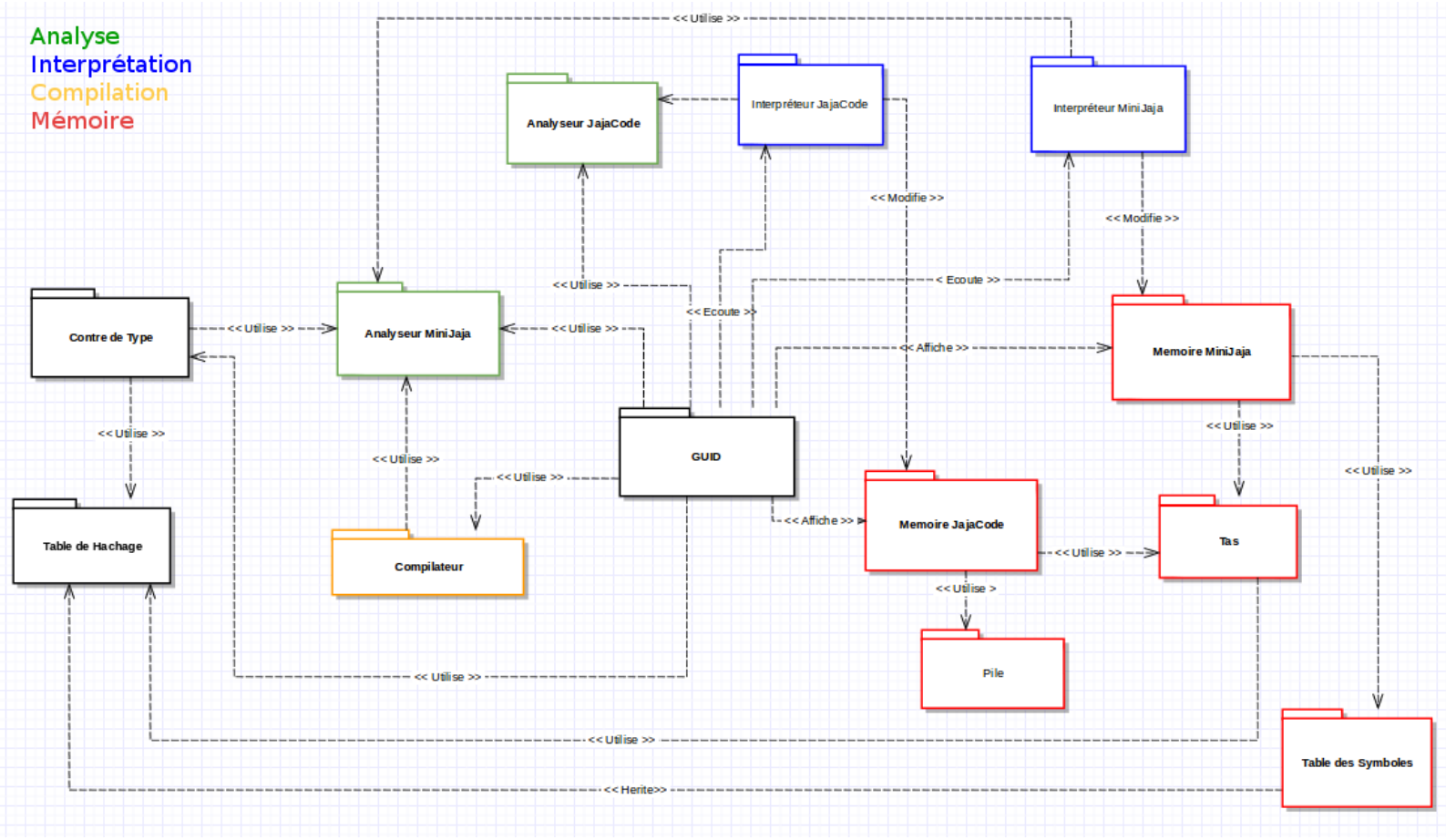
Structure du logiciel (2 minutes) :



Diapositive 1

Le diagramme de package représente tous les modules utilisé par notre projet. On en a regroupé certain en catégories (analyse, interprétation, compilation et mémoire). Il y a différentes étiquettes sur les relations entre package, car ils n'interagissent pas tous de la même manière. L'interface graphique manipule les instances.

<< Utilise >> : le module source se sert d'instances de classes du module cible comme des outils ou pour les transmettre à des objets d'autres modules (compilateur ->analyseur MiniJaja)

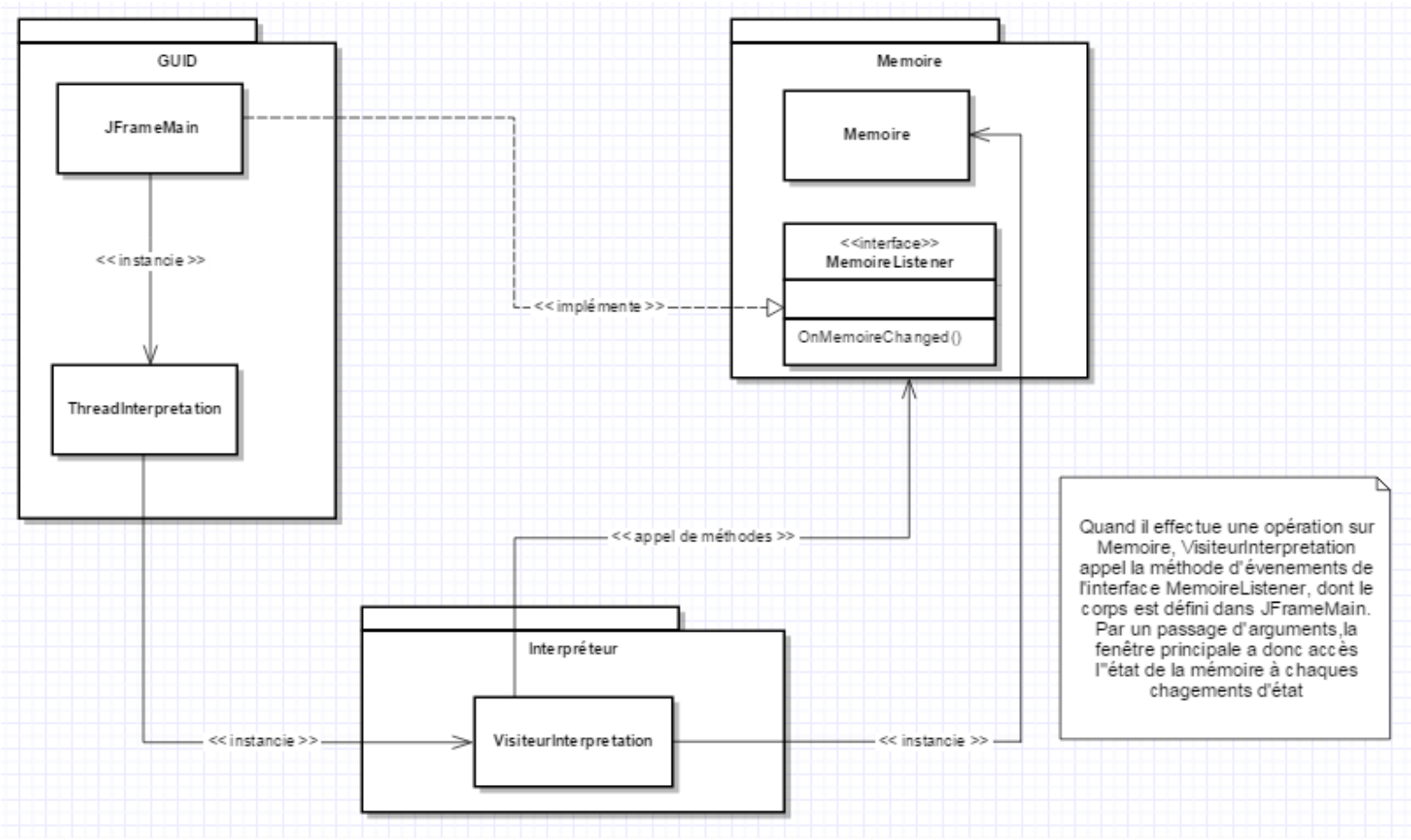
<< Affiche >> : le module source affiche l'état d'objets du module cible (guid → mémoire)

<< Herite >> : une classe du module source hérite d'une classe du module cible (table des symbole → table de hachage)

<< Ecoute >> : le module source implémente un Listener qui sera passé à des objets du module cible (guid->interpréteurs)

<< Modifie >> : le module source modifie l'état d'objets du module cible par des appels de méthodes (interpréteurs->mémoire)

Les liens se font soit par des passages de références grâce à des accesseurs ou par l'utilisation de Listener.



Diapositive 2

Les Listener sont des interfaces qui permettent la communication entre deux modules. Ils ont servi exclusivement dans le module de l'interface graphique.

La fenêtre principale implémente l'interface MemoireListener du module Memoire. Elle doit donc déclaré la fonction onMemoireChanged() prenant en paramètre un objet de type mémoire. Le constructeur du visiteur de l'interprétation prend en paramètre un MemoireListener. On peut donc lui passé la fenêtre principale en paramètre et il aura accès seulement à onMemoireChanged().

Puisqu'il modifie la mémoire à l'interprétation, à chaque opération il peut appelé onMemoireChanged(), la fenêtre peut alors appelé la méthode toString() de la classe Memoire pour l'afficher.

Interface graphique (2 minutes) :

Au début on voulais utiliser javaFX pour un meilleur rendu, mais très vite on a eut des problèmes, notamment sous Linux. Donc on est passé à Swing pour ne pas perdre de temps et produire tôt un rendu graphique.
On a utilisé l'éditeur de NetBeans pour construire graphiquement la fenêtre ce qui fais qu'on a un gros fichier pour la fenêtre principale qui contient pas mal de méthodes, mais ça a été un gain de temps.

L'interface graphique à également servie de support pour tester des modules métiers (interprétation, compilation, mémoire, etc...) car il était facile de visualiser les résultats et d'enchaîner les tests sur différents programmes MiniJaja.

La fenêtre principale utilise des threads pour les interprétations JajaCode et MiniJaja afin de pouvoir les mettrent en pause sans stopper l'affichage.

Elle implémente l'interface ErreurListener qui est passé aux constructeurs des threads. Ils appellents les méthodes jjAccept() de leur visiteurs respectifs dans un bloc try – catch et appel la méthode onErreur() du Listener en cas d'exception.

Pour l'analyse et le contrôle de type qui ne sont pas effectuer dans des threads. Ils sont également lancé dans des blocs try – catch et remonte les erreurs par des excpetions personnalisés, afin de les afficher.

Les tests de l'interface graphique ont été effectué en boite noire, il n'y a pas de test unitaire. Les testeurs ont essayé le plus de combinaison possible dans l'enchaînement des actions afin de soulever d'éventuelles exception ou provoquer des crashs

Elle permet de placé graphiquement des points d'arrêts, ce qu'on va voir dans la démo à venir.