Alexy WICIAK
Mouâd ZOUADI
Groupe 2-2

SAE n°4.03
Programmation Système Java/Assembleur

1. **Expliquez ce qu'est une pile d’appels et comment il est utilisé en Java.**

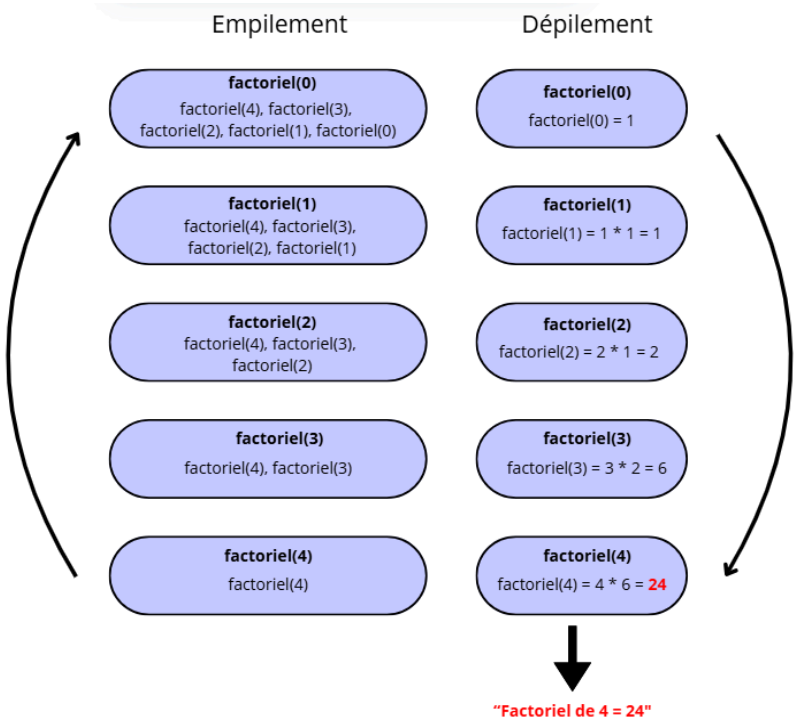
La pile d’appels est une structure de données appelée LIFO qui est utilisée par la JVM afin de suivre l’exécution des méthodes dans un ordre précis. Lorsqu’une méthode est appelée, une “stack frame” contenant les paramètres, les variables locales, l’adresse de retour et le pointeur de l’instruction (PC), est empilée.
En java, c’est le même principe mais la syntaxe est différente. Il se généralise par des appelles de méthodes. Ensuite les méthodes s'exécutent avec la JVM, ce qui modifie l’état de la pile en conséquence. Enfin, le retour de la méthode permet de la retirer de la pile, et l'exécution reprendra à l’adresse de retour stockée dans le cadre (exemple dans l’exo 1 de notre rendu).
Vous pouvez tester l’affichage de nos fonctions Java en exécutant le programme via le lien git suivant : https://github.com/AlexyWCK/SAE-Systeme-Java-Assembleur

2. **Expliquer le concept d'un cadre de pile (stack frame) et comment il est utilisé.**

Un cadre de pile (stack frame) est une sorte de boîte qui se crée à chaque fois qu'une méthode est appelée dans un programme Java. La JVM (Java Virtual Machine) empile cette boîte sur la pile d'appels, qui sert à organiser l'exécution des méthodes dans un ordre précis.
Dans cette boîte on y retrouve : les paramètres, les variables locales, l’adresse de retour et le pointeur d’instruction.
Quand la méthode finit, son cadre est retiré de la pile, et le programme reprend à l’endroit prévu.

3. **Comment la pile d'appels traite les appels de fonction récursifs**

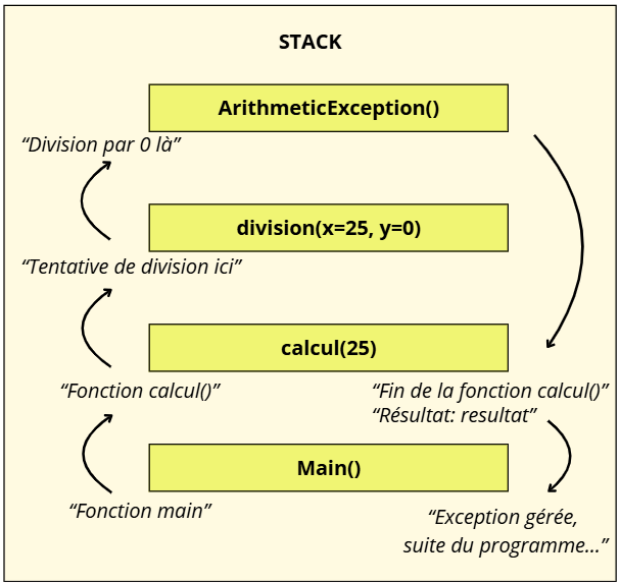
Pour comprendre comment la pile d’appels traite les appels de fonction récursifs, nous avons pris l’exemple de la fonction factorielle.
Voici le schéma que l’on a réalisé sur lequel nous nous sommes basés pour comprendre :



Dans notre programme java, nous avons illustré l'empilement des fonctions factorielles pour comprendre comment la pile fonctionne.
Lors de l'empilement, le programme ajoute les appels récursifs dans la pile jusqu'au cas de base (ici cela se produit lorsque le factoriel arrive à 0).
Ensuite, pendant le dépilement, le système résout le problème en remontant la pile, avec la formule suivante dans notre cas : n * factoriel(n - 1).
La pile d'appels permet d'exécuter les instructions dans le bon ordre et assurer la sécurité du déroulé du programme afin d'arriver au bon résultat (factoriel(4) = 24).

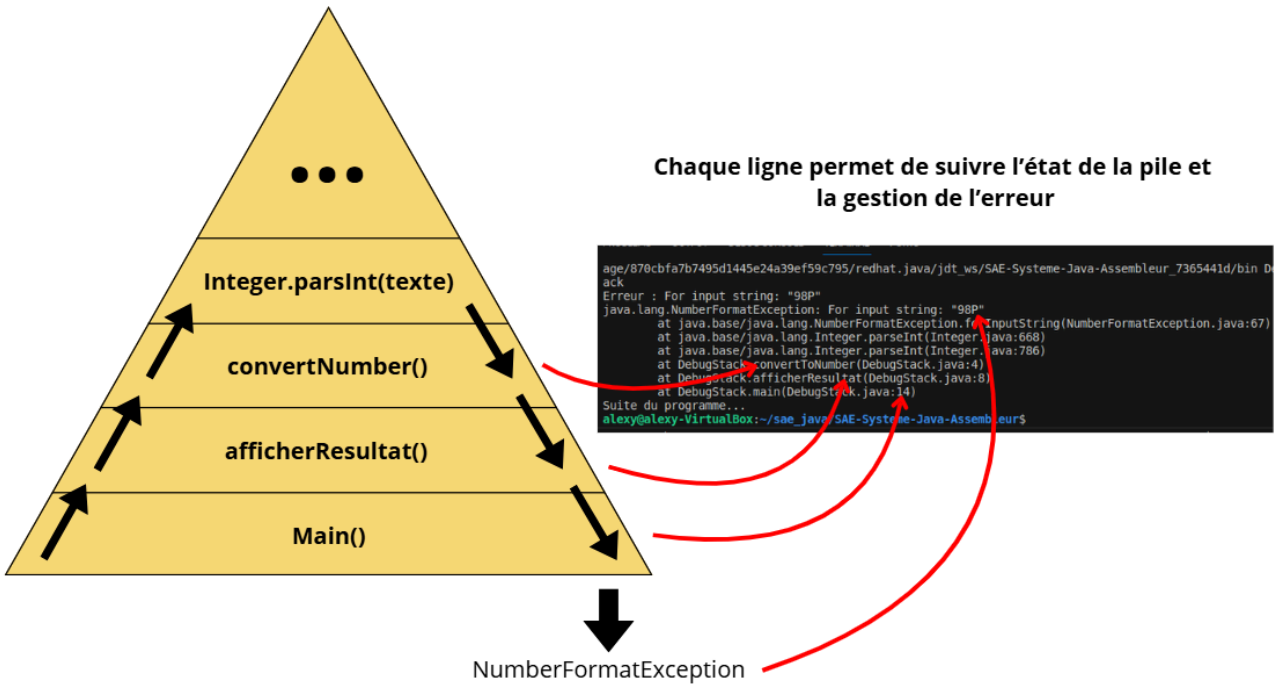
4. Comment la pile d'appels est utilisée pour gérer les exceptions en Java

Pour voir comment sont gérées les exceptions dans une pile d'appels en Java, nous avons décidé de montrer l'exemple de la division par 0 qui déclenche l'exception suivante : ArithmeticException.
En général, on utilise les instructions : "try", "catch" et "finally" pour gérer les erreurs et rendre le code plus robuste et sécurisé.
La fonction main commence par appeler la méthode calcul(25).
Cette fonction appelle la méthode division(x, y) qui essaye de diviser deux nombres passés en paramètres.
On définit le "y" à 0 pour déclencher l'exception, puis on l'affiche, et le programme continue sa progression après la gestion de l'erreur.
Nous avons mis des prints lors de chaque appel pour suivre l'état de la pile, que nous avons représenté par un schéma comme celui-ci :



5. **Comment la pile d'appels peut être utilisée pour déboguer des programmes Java**

Une des utilités principales d'étudier la pile d'appels est pour la résolution de bug lors du déroulé d'un programme. En utilisant les exceptions et des appels de fonctions cohérents, il est possible d'identifier facilement la source du problème.
Pour illustrer cet événement, nous avons décidé de prendre l'exemple d'une exception lors de la conversion d'un "str" en "int" afin de voir où se situent les erreurs qui sont gérés dans l'affichage de la pile.



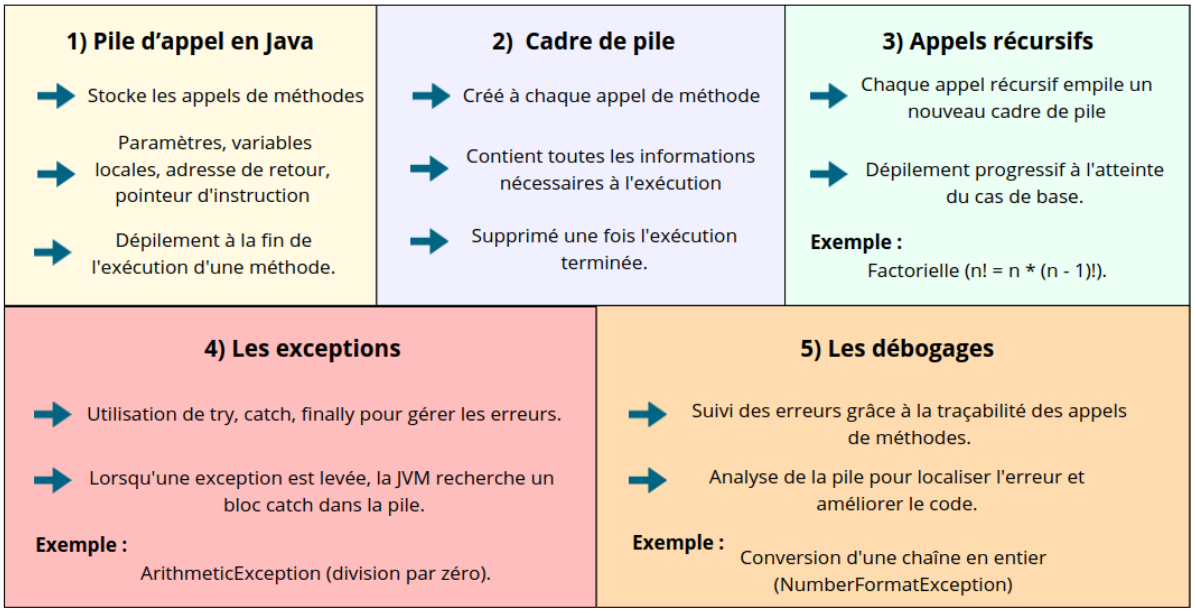
Lorsque l'on exécute ce programme, on voit dans un premier temps que l'état de la pile affiche la première erreur au niveau du main (lors de l'appel de la fonction 'afficherResultat("98P")", ligne 14). On peut ainsi suivre l'erreur pour trouver la source du

problème, qui se trouve ici à l'intérieur de cette fonction. Quand on analyse ce qu'il se passe dans l'affichage, on remarque que l'erreur provient de l'appel de la méthode : convertToNumber(). Cela nous amène enfin à la source de problème (ligne 4) qui montre que le "parsInt" ne peut pas être réalisé sur une lettre. C'est donc l'erreur la plus haute dans la pile qu'il faut prendre en compte pour résoudre les problèmes liés au code.
Ainsi, la première erreur se trouve à cet endroit, mais elle peut être gérée ailleurs (ici dans le main) avec les instructions "try", "catch", "finally" pour faciliter le traitement des erreurs.
La pile permet de suivre le traitement du problème en assurant une sécurité au niveau des méthodes et de leurs utilisations et de leurs appels.

6. Bilan: Résumez les points clés et leur importance pour comprendre et déboguer les programmes

Lors de cette SAE, nous avons pu apprendre comment la pile d'appel fonctionnait, en utilisant Java pour créer des appels de méthode. Avec certains programmes, nous avons pu apprendre à déboguer les méthodes, avec une bonne traçabilité et un suivi lors de l'affichage. En cherchant des informations et des cours sur les piles en java sur Internet, nous avons renforcé notre manière de coder en termes de sécurité et de fiabilité.
Le fait de réaliser des schémas sur le fonctionnement des pils d'appel nous a permi de comprendre les différentes étapes à réaliser pour déboguer un programme : analyser les messages d'erreur, trouver la source du problème via les appels de méthode, ajouter des affichages "system.out.println" pour savoir où en est l'exécution du programme, et le tester un maximum.
Tout ceci permet de comprendre et d'analyser les principaux problèmes auxquels nous serons amenés à être confrontés lors du développement d'applications futures.

Voici un schéma bilan :



7. Java Annexe

https://docs.oracle.com/javase/specs/jvms/se16/html/
https://www.baeldung.com/java-streams-peek-api
https://www.data-transitionnumerique.com/gestion-exceptions-java/
https://www.youtube.com/watch?v=SXhK8shduyA
https://www.lix.polytechnique.fr/~bournez/ENSEIGNEMENTS/uploads/Main/pilesfiles