



LOG2410 – Conception Logiciel

Travail personnel

Hiver 2020

Groupe [1]

Kim, Victor - 1954607

Soumis à :

François Guibault

[20/04/2020]

INTRODUCTION

Depuis l'arrivée de l'ère digitale et internet, les machines ont commencés à s'impliquer de plus en plus dans la vie quotidienne des êtres humains. On utilise de plus en plus les machines pour faciliter ou accomplir des tâches complexes. Aujourd'hui, l'ordinateur est la machine la plus utilisée par les êtres humains. Le domaine informatique est un domaine très exploré pour ces accomplissements, car c'est où qu'on va créer et donner des ordres aux machines. Les recherches dans ce domaine poussent la progression des ordinateurs. Dans le domaine de l'informatique, deux langages utilisés pour la programmation laissent des traces importantes dans l'histoire des ordinateurs. Le langage de programmation C++ et le langage de programmation java sont deux langages de logiciels très connus dans le monde digital.

Ces deux langages sont encore les langages les plus utilisés de nos jours et ils ont chacun connus des évolutions différentes au cours des années ce qui a changé plusieurs de leurs aspects. Est-ce que les deux langages de programmation en question se rapprochent ou s'éloignent ? Il faut d'abord étudier l'histoire de chacune et comparer leurs ressemblances et leurs différences pour pouvoir tirer une conclusion fiable.

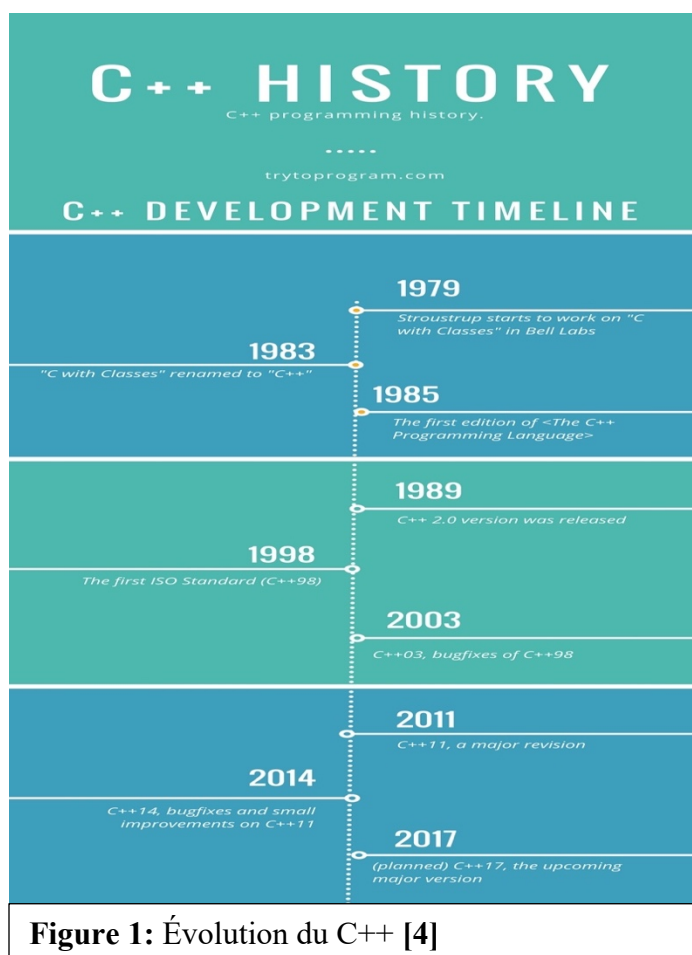
DÉVELOPPEMENT

1. Histoire du C++

Pour l'histoire du langage de programmation C++, il a été créé dans les années 1979 par l'informaticien Bjarne Stroustrup [1]. Pour sa thèse de doctorat, il a travaillé avec le langage de programmation Simula 67 qui était un langage créé pour faire des simulations.

Pendant ce temps, Simula 67 était le premier langage de programmation conçu pour pouvoir supporter les concepts de la programmation orientée objet. Stroustrup trouvait que les concepts de la programmation orientée objet étaient très utiles pour la conception des logiciels, mais que malheureusement le temps de compilation du Simula 67 était trop lent pour être utilisé en pratique dans la vie courante et que ça ne pouvait pas être utile.

En travaillant avec les classes du langage C, des entités avec des méthodes et des attributs qu'on pouvait modifier, il voulait présenter les propriétés et concepts de programmation orientée objet pour ce langage sans perdre les qualités du langage comme la vitesse [2]. En 1983, plusieurs nouveautés comme les fonctions virtuelles qui peuvent être utilisé par les classes dérivées, la surcharge des fonctions, le symbole « & » qui est utilisé pour les références par adressage, le mot « const » pour qu'une variable ou une méthode ne soit pas modifié etc. ont été ajoutés. En 1989, il y a eu l'implémentation des membres protégés « protected » pour limiter l'accès des classes et les membres statiques « static » pour étendre l'accès à toutes les composantes. En 1990, le compilateur de Turbo est présenté dans les produits commerciaux. En 1998, ils vont introduire le « Standart Template Library » qui permet aux programmeurs d'avoir accès à plusieurs algorithmes, conteneurs et itérateurs pour simplifier leur travail [3]. Il y a également eu plusieurs changements mineurs par la suite et même jusqu'à aujourd'hui le langage C++ continu son développement.



2. Histoire de Java

Pour l'histoire du langage de programmation de java, le langage a été créer en 1991 par James Gosling, Mike Sheridan et Patrick Naughton. Pendant cette période, une équipe d'ingénieur

cherchait à créer un langage de programmation qui pouvait s'adapter aux petits appareils électroniques. En s'inspirant du langage de programmation C++, ils ont conçu un langage prototype pour une télécommande digitale avec un petit écran. Cette télécommande pouvait contrôler l'audio et la vidéo. Ce nouveau langage complètement différent de ce qu'ils utilisaient auparavant n'utilise pas le même processeur. James Gosling va nommer ce langage prototype Oak. [5] Au cours des années qui suivent, avec l'implication des câblo-opérateurs dans ce projet, Oak prendra le nom de FirstPerson. Avec l'apparition du protocole http et du navigateur Mosaic, James Gosling va présenter le navigateur WebRunner l'information html associé avec des applets (des petits logiciels qui s'exécute dans la fenêtre d'une autre application). Suite à ce succès, WebRunner va être connu sous le nom de HotJava. Après quelques années, ce langage prendra enfin le nom de Java qui est le nom de l'île où une partie de la production du café est fait, car c'était le favori des programmeurs. Après plusieurs problèmes financières, java avait peu évolué aux cours de plusieurs années. La majorité des changements dans ce langage de programmation est dans les bibliothèques standards où il y a eu des modifications et ajouts de fonctionnalités afin de facilité son avancement [6]. Un des points forts du langage java étaient qu'il pouvait s'adapter à tous les machines et à tous les OS. On peut donc faire des modifications sur une classe sans avoir à recompiler les classes qui utilisent la classe modifiée [7].

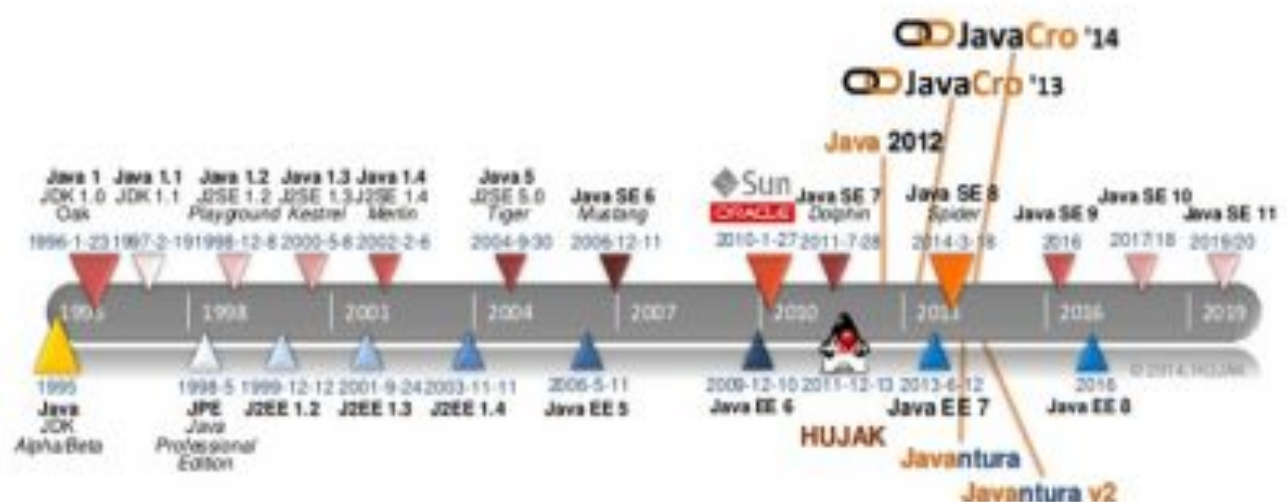


Figure 2 : Évolution des versions de Java au fil des années [8]

3. Gestion de la mémoire C++

Pour l'allocation dynamique en C++, c'est fait manuellement par les programmeurs, l'espace mémoire allouée dynamiquement va être réservée dans la partie appelée « Heap » de la mémoire. On utilise le mot-clé « new » suivi du type pour faire une demande d'allocation dynamique et s'il y a assez de mémoire, l'adresse de cette espace réservée dans la mémoire va être retournée. Il est également possible d'allouer des tableaux d'objets ou de primitives en précisant le nombre d'objets maximum qu'on veut mettre dans le tableau. Les objets, les pointeurs et les variables temporaires comme les variables locales et les variables non-statiques vont être allouées dans la partie « stack » de la mémoire [9]. Les variables primitives sont également allouées dans la partie « stack » aussi. Toutes les variables allouées dans la partie « stack » sont détruites à la sortie de la fonction où elles ont été créées. Les objets alloués dynamiquement sont alloués dans la partie « heap », pour les allocations de mémoire de cette partie, le programmeur doit désallouer manuellement l'espace mémoire en question avec le mot-clé « delete » suivi du nom de la variable, car toutes les composantes dans cette partie sont accédées avec des pointeurs, alors ils ne peuvent pas être détruite normalement.

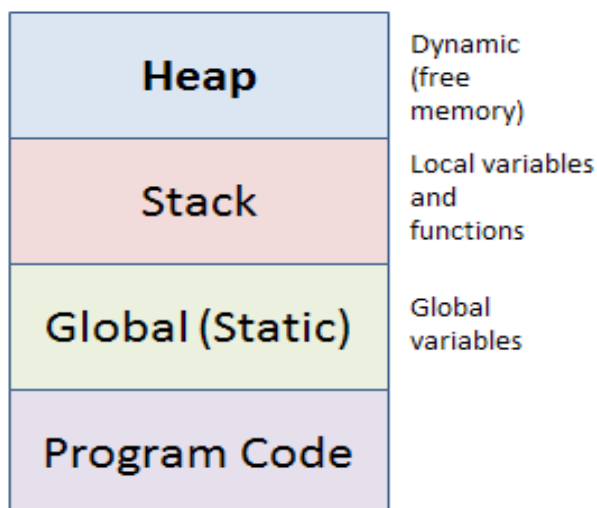


Figure 3 : Mémoire en C++ [11]

Le programmeur va donc devoir effacer manuellement l'objet dans la partie « heap » avec un « delete » et à la sortie de la fonction, le pointeur va être détruit. Pour les tableaux dynamiques

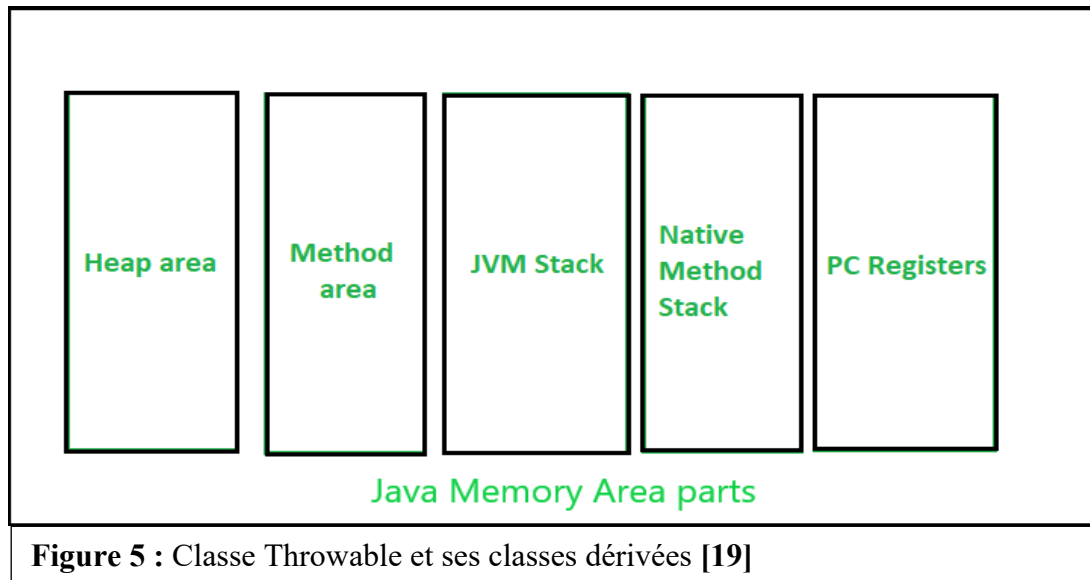
d'objets, il faudra désallouer tous les objets à l'intérieur dans une boucle et par la suite détruire le tableau. L'avantage est que le programmeur a le control total sur l'espace mémoire, il peut savoir exactement quand et où la mémoire est allouée et désallouée.

Un désavantage de la manipulation manuelle de l'espace mémoire est que si une espace mémoire n'est pas désallouée, elle va toujours restée allouée et cela va causer des fuites de mémoire. Quand les fuites de mémoire sont accumulées, cela peut faire planter le programme [10]. Une solution est d'utiliser les pointeurs intelligents qui sont un peu similaires au système de recyclage dans Java, donc l'espace mémoire va être désallouée quand les objets alloués dynamiquement ne sont plus utilisés.

4. Gestion de la mémoire en java

En java, la désallocation de l'espace mémoire se fait automatiquement, donc le programmeur n'a pas besoin de toujours vérifier s'il y a de l'espace mémoire encore allouée. Pour ce processus automatique, java utilise le system « garbage collector », ce système va désallouer automatiquement tout l'espace mémoire qu'on a utilisé dans le programme. La gestion de mémoire en Java est divisée en deux parties, le « JVM Memory Structure » et le « Working of the Garbage Collector ». La machine virtuelle de Java (JVM) va créer plusieurs espaces d'exécutions de programmes dans la partie « Heap » et ces parties sont effacées quand le système va juger qu'il n'y a plus de référencement à un objet. Ces objets vont être effacées pour libérer la mémoire [12]. De plus, la machine virtuelle de Java contrôle le système de « Garbage Collector », donc quand il va juger que la mémoire est insuffisante, ce système va être appelé. Un des meilleurs avantages de java est qu'il n'y a pas de fuites de mémoire, car tous les objets vont toujours être effacés à la fin. Un petit désavantage est que si le programmeur veut utiliser un objet qu'il avait créé auparavant, il se peut que le système ait déjà désalloué l'espace mémoire de l'objet. Dans la mémoire en java, il y a une espace pour les méthodes. Cette espace va contenir la structure des classes, le nom des super classes, le nom d'interface et les constructeurs. Il y a également l'espace « Heap » qui va contenir les objets que le programmeur a créé et une partie « Stack » qui va contenir les références aux objets et les primitives [13]. Donc la différence principale est

que tous les objets en Java sont sauvegardés dans la partie « heap », car tous les objets sont alloués dynamiquement et que seulement les primitives et les référencements aux objets sont sauvegarder dans la « stack ».



5. Gestion des exceptions en C++

Les exceptions sont des problèmes informatiques qui arrivent durant l'exécution d'un programme. En C++, les exceptions sont gérées avec l'utilisation de trois mots-clés, « throw », « catch » et « try ». En premier, le bloc qui contient le mot-clé « try » va trouver les blocs de code avec des exceptions particulières. Après avoir détecté les exceptions pendant l'exécution, il va lancer cette exception en utilisant le mot-clé « throw ».

Par la suite, un autre bloc va attraper l'exception lancé et le gérer en utilisant le mot-clé « catch » [15]. Comme on peut voir dans le code en bas, lorsque le dénominateur est 0 le « try » va le détecter pendant l'exécution du programme et va lancer cette exception. Par la suite, le bon block « catch » va attraper cette exception et la gérer. C++ va gérer ces exceptions en donnant le contrôle à des fonctions faites pour gérer ces exceptions qui sont appelées les « handlers ». En C++, on peut jeter tous les types comme exception, même les primitives et les pointeurs [17]. On peut également utiliser la librairie « exception » qui contient la classe de base de toutes les exceptions et plusieurs outils pour aider aux gestions d'erreurs [18]. Il y a aussi le block « catch(...) » qui permet d'attraper toutes les types d'exceptions.

```
8
9 #include <iostream>
10
11 using namespace std;
12
13 int main() {
14
15     double numerateur=1;
16     double denominateur=0;
17
18     try {
19         if(denominateur==0) {
20             throw denominateur;
21         }
22     }
23     catch(double exception) {
24         cout <<"division par 0" << endl;
25     }
26
27     double result=numerateur/denominateur;
28     cout << result << endl;
29
30
31     return 0;
32 }
```

division par 0
inf
Program ended with exit code: 0

[16] capture d'écran de Victor Kim

Les erreurs ne sont pas des exceptions, car ils ne peuvent pas être gérées. Il existe deux types d'erreurs, les « logic errors » qui sont les erreurs faites par les programmeurs pendant la programmation et les « runtime errors » qui sont les erreurs externes que les programmeurs ne peuvent pas contrôler.

6. Gestion des exceptions en Java

En java, toutes les erreurs et les exceptions sont des classes dérivées de la classe « Throwable ». Une branche contient la classe « Exceptions » et ses dérivées. L'autre branche contient la branche « Error » et ses dérivées. En Java, on peut seulement jeter les objets qui sont des dérivées de la classe « Throwable ». La classe « Exceptions » et ses sous-classes contiennent les conditions que le programme d'utilisateur doit attraper et la classe « Error » et ses sous-classes contiennent les erreurs qui ont un lien avec le JRE qui ne doit pas essayer d'attraper. Pour gérer

une exception par défaut quand il a une exception qui se produit dans une méthode, la méthode va créer un objet de type « Exception » et va le lancer au system d'exécution (JRE). L'objet va contenir l'information sur l'exception et sur l'état du programme où l'exception s'est produite. Une série de méthode vont être appelé pour se rendre à la place où l'exception a été produite. La série de méthode appelée est la « stack ». Le système va parcourir le « stack » en ordre inverse pour essayer de trouver s'il y a un bloc de code qui sera capable de gérer l'exception si aucune méthode peut gérer l'exception, le système va le donner au « default exception handler » qui va terminer le programme. Les blocs de code pour gérer les exceptions sont composées avec les même composantes qu'en C++, donc un « try », un « throw » et un « catch » mais aussi d'un bloc « finally » qui va exécuter un morceau de code même s'il y a des exceptions qui n'ont pas été gérées [20]. La gestion des exceptions pour les deux sont assez similaires dans le mécanisme de lancer les exceptions et que le bon programme vient gérer l'exception avec quelques différences au niveau de l'implémentation, au niveau des restrictions et au niveau des extensions. En C++, on peut lancer tous les types comme une exception tandis qu'en Java les types qui ne dérivent pas de « Throwable » ne peuvent pas être lancer. Le bloc « catch (...) » propre a C++ ou le bloc « Finally » propre a Java.

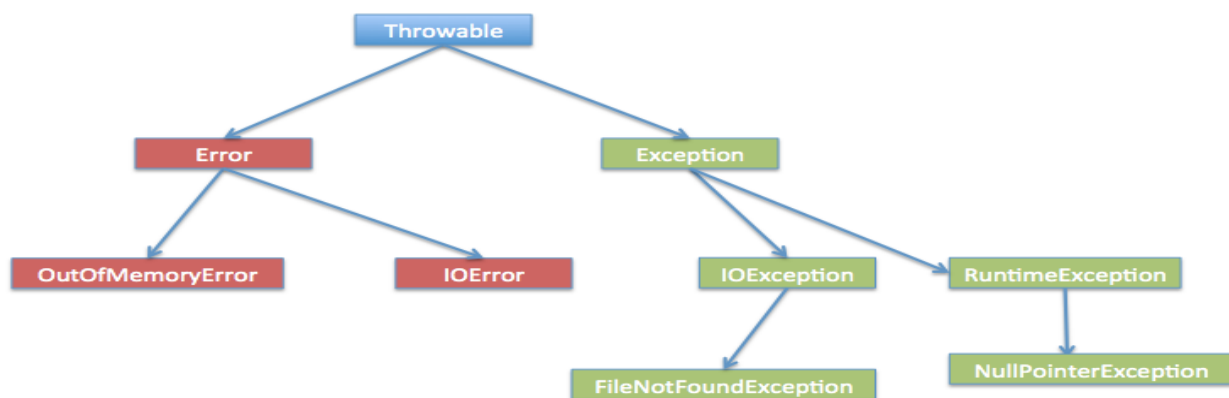



Figure 4 : Composantes dans la mémoire JVM [14]

7. Les types génériques en C++

La programmation générique est une façon de coder des méthodes, classes et interfaces de sorte qu'on n'a pas besoin de spécifier clairement le type. Cela rend les manipulations plus faciles, car on peut implémenter du code qui fonctionne pour tous les types, donc on peut réutiliser le code. On utilise les « templates » pour l'implémentation des types génériques, on les utilise pour les classes ou fonctions qui ne dépendent pas des types par exemple, les conteneurs qu'on va utiliser pour sauvegarder des données [22].

```
9  #include <iostream>
10
11  using namespace std;
12
13  template<typename T>
14  void print(T object) {
15      cout << object << endl;
16  }
17
18  int main() {
19
20      int a=8;
21      string b="chaîne de caractere";
22
23      print(a);
24      print(b);
25  }
```



```
8
chaîne de caractere
Program ended with exit code: 0
```

[21] Capture d'écran de Victor Kim

On peut voir dans la photo au-dessus, pour que la fonction « print() » soit de type générique, il a simplement suffit de mettre « template<typename T> » dont T peut être n'importe quel lettre. Pour les variables d'entrées, il suffit simplement de mettre le « T » devant pour que le programme fait les manipulations sur les variables d'entrées selon leur type d'origine. On voit que les 2 variables sont imprimées correctement même s'ils n'ont pas le même type. En C++, à chaque rencontre de types génériques, le compilateur va créer un fonction propre au type dont on lui passe en paramètre.

8. Les types génériques en Java

Donc les types génériques ont tous le même but, de permettre la réutilisation du code, de permettre l'implémentation des classes, des méthodes et des interfaces sans types spécifiques. Une différence avec le C++ est l'implémentation, en java on écrit directement à côté du nom de la classe « <T> » dont la lettre permet de préciser qu'il ne s'agit pas d'un type et qu'il faut manipuler la composante selon ce que le programmeur va le donner. Dans **Figure 6**, on peut voir que la méthode « `isEqual()` » et la méthode « `compare()` » sont des méthodes de types génériques, donc ils peuvent prendre n'importe quel type de variables passées en paramètres. Une autre différence est qu'en Java, après la compilation, le compilateur va remplacer toutes les références à des types génériques avec les « cast » nécessaires pour obtenir la bonne conversion, contrairement au C++ qui va créer plusieurs versions pour une composante durant l'exécution. On peut voir que les différences principales sont au niveau de l'implémentation du code et au niveau du mécanisme utilisé pour avoir les résultats attendus avec les types génériques.

```
public class GenericsMethods {  
  
    //Generics in method  
    public static <T> boolean isEqual(GenType<T> g1, GenType<T> g2){  
        return g1.get().equals(g2.get());  
    }  
  
    public static <T extends Comparable<T>> int compare(T t1, T t2){  
        return t1.compareTo(t2);  
    }  
}
```

Figure 6 : Les méthodes avec types génériques [23]

Conclusion et point de vue personnel

En conclusion, on peut voir que Java était une langue de programmation créée en s'inspirant de la langue de programmation C++. La gestion de mémoire où tous les objets en Java sont alloués

dynamiquement et les objets normaux en C++ sont alloués sur la « Stack ». La gestion des exceptions où C++ permet de jeter tous les types comme une exception tandis qu'en Java seulement les objets des classes dérivées de « Throwable » sont lancés. Les types génériques où le compilateur en C++ va créer une nouvelle version à chaque nouveau type contrairement au compilateur Java qui va juste effacer tous les types génériques et mettre les conversions nécessaires. Tous les aspects ont un principe plus ou moins similaire avec des différences qui sont principalement au niveau de l'implémentation ou du mécanisme pour permettre de faciliter des processus, mais le produit final reste le même. Il y a également plusieurs autres différences mineures dans les aspects abordés, mais on peut voir que dans ces points majeurs, on peut quand même ressortir assez de similitudes.

Selon moi, je pense qu'avec l'évolution des deux langages de programmation, elles deviennent de plus en plus proche l'un de l'autre, car les deux langues ont fondamentalement une origine très similaire, Java est une langue de programmation créée en s'inspirant de la langue C++ et que les différences marquantes sont que Java n'a pas de pointeurs et que la désallocation des objets alloués dynamiquement est faite automatiquement. Même les différences au niveau de l'implémentation ou au niveau du mécanisme, le résultat obtenu est la même en C++ et en Java. Les concepts sont également très similaires, par exemple, la désallocation automatique de la mémoire en Java est très semblable aux pointeurs intelligents en C++, les deux ont l'utilité de faire le ménage et réduire les fuites de mémoire. Donc même après plusieurs années d'évolution distinctes, la ressemblance des utilités est toujours présente, car les deux langages évoluent toujours dans la même direction donc la différence va souvent être au niveau de l'implémentation du code ou du mécanisme, mais la logique est souvent très similaire ou même identique. Tout ce qu'on peut accomplir en C++, on peut également le faire en Java d'une autre façon et cette façon va avoir des éléments communs avec le C++. Clairement, Java n'est pas une langue complètement différente avec des extensions spéciales qui permet d'accomplir des choses uniques.

Références

- [1] Pramanick S. (2019). History of C++, [En ligne]
Disponible : <https://www.geeksforgeeks.org/history-of-c/>
- [2] TRYTOPROGRAM. (2020). Who developed C++, [En ligne]
Disponible : <http://www.trytoprogram.com/cplusplus-programming/history/>
- [3] Polytechnique. (2020). Early C++, [en ligne] Disponible :
<http://www.enseignement.polytechnique.fr/informatique/INF478/docs/Cpp/en/cpp/language/history.html>
- [4] TRYTOPROGRAM. Who developed C++, [En ligne]
Disponible : <http://www.trytoprogram.com/cplusplus-programming/history/>
- [5] MathBits. (2020). Brief History of Java, [En ligne] Disponible :
<https://mathbits.com/MathBits/Java/Introduction/BriefHistory.htm>
- [6] Pierre B. (2018). Le langage Java : histoire, caractéristiques & popularité, [En ligne]
Disponible : <https://www.silkhom.com/langage-java-histoire-caracteristiques-popularite/>
- [7] Engel J. (2003). Java Binary Compatibility, [En ligne] Disponible :
<https://www.informit.com/articles/article.aspx?p=32088>
- [8] Pierre B. (2018). Évolution des versions de Java au fil des années, [En ligne] Disponible :
<https://www.silkhom.com/langage-java-histoire-caracteristiques-popularite/>
- [9] GeeksForGeeks. (2018). New and delete operators in C++, [En ligne] Disponible :
<https://www.geeksforgeeks.org/new-and-delete-operators-in-cpp-for-dynamic-memory/>
- [10] Hinkelmann F. (2017). Confused about stack and heap?, [En ligne] Disponible :
<https://medium.com/fhinkel/confused-about-stack-and-heap-2cf3e6adb771>
- [11] Gibbs M. Memory in C++, [En ligne] Disponible :
<https://study.com/academy/lesson/how-to-allocate-deallocate-memory-in-c-programming.html>
- [12] Vaidya A. (2018). Garbage Collector in Java, [En ligne] Disponible :
<https://www.geeksforgeeks.org/garbage-collection-java/>
- [13] JavaTPoint. (2018). Memory Management in Java, [En ligne] Disponible :
<https://www.javatpoint.com/memory-management-in-java>
- [14] GeeksForGeeks. (2018). Java Memory Management, [En ligne] Disponible :

<https://www.geeksforgeeks.org/java-memory-management/>

[15] GeeksForGeeks. (2018). Exception Handling in C++, [En ligne] Disponible :

<https://www.geeksforgeeks.org/exception-handling-c/>

[17] tutorialpoint. (2016). C++ Exeption Handling, [En ligne] Disponible :

https://www.tutorialspoint.com/cplusplus/cpp_exceptions_handling.htm

[18] cplusplus. Standart exception, [En ligne] Disponible :

<http://www.cplusplus.com/reference/exception/>

[19] JavaCodeGeeks. Java Exception Handling, [En ligne] Disponible :

<https://www.javacodegeeks.com/2013/07/java-exception-handling-tutorial-with-examples-and-best-practices.html>

[20] Rollbar. Throwing exceptions in Java, [En ligne] Disponible :

<https://rollbar.com/guides/java-throwing-exceptions/>

[22] Webb C. (2017). Generics With C++, [En ligne] Disponible :

<https://medium.com/journey-of-one-thousand-apps/generics-in-c-5b0354786b54>

[23] Fadatare R. (2018). Java Generic Methods Examples, [En ligne] Disponible :

<https://www.javaguides.net/2018/08/java-generic-methods-examples.html>