Petit comparatif Java vs C++

Alexandre Patry

6 mars 2006

Table des matières

1 Objectifs

L'étudiant devra pouvoir :

1. Déclarer des variables et des constantes (peut-être enum)
2. Coder des séquences d'instructions
3. Connaître les structures de contrôles (conditionnelle et boucles)
4. Pouvoir créer des programmes (connaître fonction main)
5. Déclarer, implémenter et appeler des fonctions
6. Déclarer et utiliser des tableaux
7. Déclarer et utiliser des chaînes de caractères c et c++
8. Utiliser les flux de fichiers et d'entrée sortie
9. Appeler des fonctions standards (include)
10. Pouvoir compiler des programmes

2 Généralités

Tout comme en `java`, en `c++` les instructions sont terminées par des `;`.

3 Variables scalaires

3.1 Utilisation

La déclaration de variable primitives en `java` et en `c++` sont très similaires et suivent le modèle suivant : `<type> <nom> = <valeur initiale>`, où l'assignation à une valeur initiale est facultative. Alors que la déclaration d'une variable constante est préfixée du mot clé `final` en `java`, elle est préfixée du mot clé `const` en `c++`.

Cependant, différemment de `java`, `c++` n'assigne aucune valeur par défaut aux variables non-initialisées.

Pour des exemples dans chaque langage, voyez les figures 2 et 3.

	`c++`	`java`
Type booléen	`bool`	`boolean`
Types entiers	`char`, `wchar_t`, `short int`, `int`, `long int`	`byte`, `char`, `short`, `int`, `long`
Types réels	`float`, `double`, `long double`	`float`, `double`
Type *vide* `void`	`void`	

FIG. 1 – Types primitifs en `c++` et en `java`.

```
char c = 'a'; // Déclare un variable nommée 'c' contenant
              // le charactère 'a'

const int i = 1; // Déclare un entier nommé 'i' et
                 // l'initialise à 1

double d;  // Déclare un double nommé 'd' contenant une
           // valeur non-définie
```

FIG. 2 – Déclaration de variables scalaires en `c++`

```
char c = 'a'; // Déclare un variable nommée 'c' contenant
              // le charactère 'a'

final int i = 1; // Déclare un entier nommé 'i' et
                 // l'initialisé à 1

double d;  // Déclare un double nommé 'd' et l'initialise
           // par défaut à 0.0
```

FIG. 3 – Déclaration de variables scalaires en `java`. Voir la figure 2 pour l'équivalent `c++`.

3.2 Détails techniques

Vous trouverez dans la figure 1 les différents types fondamentaux de `java` et `c++`. Comme `java` est un langage récent, son type `char` est assez grands (deux octets) pour encoder les lettre d'à peu près tout les alphabets utilisés. Cependant, le type `char` de `c++`, tenant habituellement sur 1 octet, ne permet d'encoder que les charactères de bases (souvent seulement les charactères latins). Pour palier à cette lacune, le type `wchar_t` a été introduit, pouvant lui exprimer un jeu de charactères étendus.

Cependant, même si `wchar_t` semble plus adapté que `char` pour gérer des textes, pour des raisons historiques, le type `char` est tout de même celui qui est le plus utilisé et est probablement celui qui vous intéressera.

Alors qu'en `java`, la taille (nombre d'octets) de chaque type est standardisée, en `c++` la norme est plus souple. En effet, la taille des différents types dépendra souvent de l'architecture sous-jacente (e.g. les entiers auront une taille naturelle pour le processeur, soit 32 bits sur une machine intel standard ou 64 bits sur un AMD64).

4 Tableaux

Les tableaux permettent de rassembler plusieurs variables d'un même type dans des espaces mémoires adjacents.

4.1 Utilisation

La syntaxe pour déclarer et utiliser des tableaux en `c++` et en `java` est très semblable. Cependant, les tableaux `java` et les tableaux `c++` partagent quand même quelques différences :

- En `c++`, la taille du tableau doit être connue et spécifié à la compilation.
- En `c++`, un tableau est créé automatiquement à la déclaration (il n'est pas nécessaire de faire un appel à `new` pour que le tableau soit alloué).
- `c++` vous oblige à mettre les `[]` après le noms de la variables à la déclaration (alors que `java` vous permet de les mettre aussi après le type).
- Les tableaux `c++` ne connaissent pas leur taille (pas d'attribut `length` comme en `java`), donc il faut spécifier la taille du tableau à l'aide d'une autre variable.
- Comme les tableaux `c++` ne connaissent pas leur taille, il leur est impossible de vérifier si un accès est hors bornes. C'est donc à vous d'être vigilant pour éviter les *core dump*.

consultez les figures 4 et 5 pour des exemples de déclarations et d'utilisation de tableaux.

```
char c[10];  // cré un tableau de 10 char

int i[] = {1, 2, 3, 4}; // cré un tableau de 4 int
                        // initialisés respectivement
                        // à 1, 2, 3 et 4

double d[5] = {2}; // cré un tableau de 5 double
                   // initialisés respectivement
                   // à 2, 0, 0, 0, 0

bool b [2][4];   // cré un tableau bidimensionnel de bool

i[1];        // retourne 2

i[10];       // ne fait aucune erreur, mais retourne un
             // valeur indéfinie

b[1][1] = true; // assigne l'élément (1, 1) de b à true.
```

FIG. 4 – Déclaration de tableaux en `c++`

```
char c[];  // cré une référence vers un tableau de char,
           // mais n'alloue pas les 10 char

c = new char[10];  // alloue un tableau de 10 char

int i[] = {1, 2, 3, 4}; // cré un tableau de 4 int
                        // initialisés respectivement
                        // à 1, 2, 3 et 4

// double d[5] = {2} est illégal en java

boolean b [][] = new bool[2][4]; // cré un tableau
                                 // bidimensionnel de
                                 // boolean

i[1];          // retourne 2

i[10];         // lance une exception

b[1][1] = true; // assigne l'élément (1, 1) de b à true.
```

FIG. 5 – Déclaration de tableaux en `java`. Voir la figure 4 pour l'équivalent `c++`.

5 Fonctions

5.1 Utilisation

La syntaxe des définitions de fonctions en `c++` est très semblable à celle de `java` à la différence que :

- `c++` permet de définir des fonctions à l'extérieur d'une classe.
- Pour qu'une fonction puisse être utilisée en `c++`, elle doit avoir été déclarée avant l'appel.
- `c++` vous permet de passer n'importe quel paramètre par valeur ou par référence, alors que `java` passe les objets par référence et les types fondamentaux par valeur.
  Par défaut, les paramètres sont passés par valeur (à l'exception des tableaux). Pour passer un paramètre par référence, il faut placer une `&` entre le type du paramètre et son nom.

L'appel d'une fonction se fait de la même façon en `java` et en `c++`.

Pour des exemples de déclaration de fonctions, voyez les figures 6 et 7.

6 Les bibliothèques

Les bibliothèques de fonctions et de classe permettent d'appeler du code déjà existant sans avoir à le copier dans le fichier appelant. L'équivalent `java` serait les paquetages.

6.1 Utilisation

Alors qu'en `java` l'instruction `import paquet.Module` est utilisée pour importer un module, l'instruction `#include <module>` est utilisée. Si vous voulez utiliser les librairies standards, vous devez aussi ajouter la commande `using namespace std ;` après votre dernière instruction `#include`.

Consultez les figures 8 et 9 pour des exemples d'importation de bibliothèques.

7 Chaînes de caractères

7.1 Chaînes `c`

Les chaînes de caractères en `c++` ne sont rien d'autres que des tableaux de caractères où l'élément après la dernière lettre à la valeur numérique `0` (caractère `'0'`).

Vous pouvez lancer `man string` pour connaître différentes fonctions pouvant être utilisées avec les chaînes de caractères `c`.

```
/**
 * Fonction ne prenant aucun paramètre et ne retournant
 * rien.
 */
void f (){
    /* corps de la fonction */
}

/**
 * Fonction prenant un paramètre entier par valeur et
 * retournant un double.
 */
double g (int i){
    /* corps de la fonction */
}

/**
 * Fonction prenant un paramètre entier par référence et
 * retournant un double.
 */
double h (int& i){
    /* corps de la fonction */
}

/**
 * Fonction prenant en paramètre un objet par valeur.
 * Une copie de l'objet est passé à la fonction.
 */
void i (MaClasse monObjet){
    /* corps de la fonction */
}
```

FIG. 6 – Les fonctions en c++

```java
/**
 * Classe qui contiendra les fonctions car java ne permet
 * pas les fonctions libres.
 */
public class MaClasse{
    /**
     * Fonction ne prenant aucun paramètre et ne
     * retournant rien.
     */
    public void f (){
        /* corps de la fonction */
    }

    /**
     * Fonction prenant un paramètre entier par valeur et
     * retournant un double.
     */
    public double g (int i){
        /* corps de la fonction */
    }

    /**
     * Pour passer un type primitif par référence en java,
     * il faut passer par un objet.
     */
    public double h (Integer i){
     /* corps de la fonction */
    }

    /**
     * Java ne permet pas de passer un objet par valeur.
     * Ce qui s'en rapprocherait le plus serait la
     * fonction suivante.
     */
    void i (MaClasse monObjet){
        MaClasse o = monObjet.clone ();

        /* corps de la fonction qui manipule o à la place
           de monObjet */
    }
}
```

FIG. 7 – Les fonctions en `java`. Voir la figure 6 pour l'équivalent `c++`.

```
#include <vector>
#include <string>

using namespace std;

// les chaînes et les vecteurs peuvent maintenant être
// utilisés
```

FIG. 8 – Importation de bibliothèques en `c++`

```
import java.util.Vector;
import java.lang.String;
```

FIG. 9 – Importation de bibliothèques en `java`. Voir la figure 8 pour l'équivalent `c++`.

7.2 Chaînes `c++`

Une classe ressemblant un peu plus à la représentation des chaînes de `java` est présente dans les bibliothèques standards. Pour l'utliser, assurer vous d'inclure la bibliothèque `string`. Voici quelques différences entre la classe `string` de `c++` et la classe `String` de `java` :

- Les opérateurs `< <= == >= > !=` peuvent être utilisés sur les `string c++` pour comparer leur contenu.
- Les `string c++` contiennent des caractères sur 8 bits (ASCII) alors qu'en `java` ils sont sur 16 bits (unicode).
- La classe `String` en `java` ne permet pas de changer la chaînes sous jacente, alors que la classe `string` en `c++` le permet.

8 Entrées/sorties console

8.1 Utilisation

Comme en `java`, les entrées/sorties `c++` sont abstraites par des flux. Ces flux sont cependant différent.

Les flux d'entrée/sortie console sont définis dans la bibliothèque `iostream` et se nomment `cin` (entrée standard), `cout` (sortie standard) et `cerr` (sortie pour les erreurs).

Voyez la Figure 13 pour plus de détails.

```c
#include <cstring> // pour les fonctions sur les chaînes
using namespace std;

void f () {
    char* s1 = "allo le monde"; // déclare une chaîne
    char* s2 = "wazzup";
    char s3[50];

    // comparer deux chaînes
    int comp = strcmp (s1, s2);

    if (comp < 0){
    /* s1 < s2 */
    }
    else if (comp == 0){
    /* s1 == s2 */
    }
    else if (comp > 0){
    /* s1 > s2 */
    }

    int longueur = strlen (s1); // longueur de s1

    strncpy (s3, s1, 50); // copie s1 dans s3

    // on peut modifier la copie s3, mais pas s1 ni s2
    s3[0] = 'A'; // change le premier caractère de s3
}
```

FIG. 10 – Utilisation de chaînes c

```
#include <string> // pour la classe string
using namespace std;

void f () {
    string s1 = "allo le monde"; // déclare une chaîne
    string s2 = "wazzup";
    string s3;

    // comparer deux chaînes
    if (s1 < s2){
    /* s1 < s2 */
    }
    else if (s1 == s2){
    /* s1 == s2 */
    }
    else if (s1 > s2){
    /* s1 > s2 */
    }

    int longueur = s1.size (); // longueur de s1

    s1 = s3; // copie s1 dans s3
}
```

FIG. 11 – Utilisation de chaînes c++

```
...
void f () {
    String s1 = "allo le monde"; // déclare une chaîne
    String s2 = "wazzup";
    String s3;


    // comparer deux chaînes
    int comp = s1.compareTo (s2);

    if (comp < 0){
    /* s1 < s2 */
    }
    else if (comp == 0){
    /* s1 == s2 */
    }
    else if (comp > 0){
    /* s1 > s2 */
    }

    int longueur = s1.length (); // longueur de s1

    s1 = s3; // s3 référence maintenant vers la même
             // chaîne que s1
}
```

FIG. 12 – Utilisation de chaînes `java`. Voir la figure 11 pour l'équivalent `c++`.

```
#include <iostream> // pour utiliser cin, cout et cerr
using namespace std;

...

// lire l'entrée standard caractère par caractère
for (char c = cin.get (); cin.good (); c = cin.get ())
{
    // traiter c
}

...

// lire l'entrée standard mot par mot (blancs ignorés)
string mot;
while (cin >> mot)
{
    // trainter mot
}

...

// lire l'entrée standard ligne par ligne (sans '\\n')
string ligne;
while (getline (cin, ligne))
{
    // traiter ligne
}

...

// écrire sur la sortie standard (endl pour finir ligne
// et vider tampon)
cout << "Chaine suivi d'un entier " << 3 << endl;

...
```

FIG. 13 – Utilisation de flux en c++

```
#include <fstream> // pour utiliser les flux de fichiers
using namespace std;

...

// déclarer un flux d'entrée
ifstream in;

in.open ("fichier.in");

// utiliser in comme cin

...

// déclarer un flux de sortie
ofstream out;

out.open ("fichier.out");

// utiliser out comme cout
...
```

FIG. 14 – Utilisation de flux en `c++`

9 Entrées/sorties fichiers

9.1 Utilisation

Les flux de fichiers s'utilisent comme les flux d'entrée/sortie de console. Ils sont définis dans la bibliothèque `fstream`. Pour un exemple voyez 14.

10 Classes

Tout comme `java`, `c++` permet aussi de faire de la programmation orientée objet. Cependant, les deux langages se différentient beaucoup quant à leur réalisation de ce concept.

10.1 Utilisation

Voici les différences principales entre une classe `c++` et une class `java`

- Une classe `c++` est publique et n'a pas de qualificateur (`public, protected, private`) avant sa définition.
- Alors qu'en `java`, le mot clé `extends` permet d'hériter d'une classe, en `c++` le `:` est utilisé. De plus, on doit spécifier si l'héritage est `public`, `private`

ou `protected`.
- `c++` permet à une classe d'avoir plus d'une classe de base.
- En `java`, les méthodes sont virtuels par défaut, alors qu'en `c++` pour l'être, elles doivent être préfixés du mot clé `virtual`.
- En `c++`, il est possible de spécifier qu'une méthode ne modifie pas l'état de l'objet à l'aide du mot clé `const` placé tout de suite après la parenthèse fermante de la déclaration.
- En `c++`, il n'y a pas d'interface à proprement dit. Cependant, pour signifier qu'une méthode est *abstraite*, elle doit être précédé du mot clé `virtual` et `= 0` doit précéder le `;` de sa définition (e.g. `virtual void f () = 0 ;`). Donc, une classe seulement composée de méthode abstraites est équivalente à une interface.
- En `c++` il n'y a pas de classe parente commune à toutes les classes, alors qu'en `java` il y a la classe `Object`.

11 Gestion des exceptions

11.1 Usages

Comme `java`, `c++` permet de gérer les erreurs avec des exceptions. Cependant, les deux langages ont les différences suivants :

- Alors qu'en `java` seulement les objets ayant pour parent la classe `Exception` peuvent être lancées, en `c++` n'importe quoi peut être lancé (objet ou type fondamental).
- Il n'y a pas de blocs `finally` en `c++`, mais leur comportement peut être simulé avec un destructeur.
- Lorsqu'une exception est lancée en `c++`, tous les objets devenant hors de porté sont détruit et leur destructeur est appelé.
- Alors que `java` oblige toutes les opérations pouvant lancer des exceptions à être dans des `try catch`, `c++` ne l'oblige pas.
- Une clause `catch` attrapant toutes les exceptions a la forme `catch (...)`.
- Le mot clé `c++ throw` sert à la fois pour deux mots clés `java throw` et `throws`.
- En `c++`, ce n'est pas obligatoire de mettre dans la signature d'une fonction les exceptions qu'elle peut lancer. Cependant, si des exceptions sont spécifiées, cela signifie que seulement ceux-là peuvent être lancées par la fonction.

12 Pointeurs et gestion de mémoire

12.1 Pointeurs

Les pointeurs permettent de référencer des espaces mémoires sans les occuper.

```
/** Déclaration de la classe */
class MaClasse : public ClasseMere, public MonInterface
{
public :
    // Début section publique

    /** Constructeur par défaut. */
    MaClasse ();

    /** Destructeur. */
    virtual ~MaClasse ();

    /** Fonction publique */
    void f ();

private :
    // Début section privée
    void g ();

    int m_i;
};

/** Implémentation de la classe. */
MaClasse::MaClasse ()
{
    /* votre code ici */
}

MaClasse::~MaClasse ()
{
    /* votre code ici */
}

void MaClasse::f ()
{
    /* votre code ici */
}

void MaClasse::g ()
{
    /* votre code ici */
}
```

FIG. 15 – Une classe en c++.

```java
public class MaClasse
    extends ClasseMere
    implements MonInterface
{
    /** Constructeur par défaut. */
    public MaClasse ()
    {
        /* implémentation */
    }

    // Il n'y a pas de destructeur en java, seulement la
    // méthode finalize, mais elle n'est pas certaine
    // d'être appelée.

    /** Méthode publique. */
    public void f ()
    {
        /* implémentation */
    }

    /** Méthode privée */
    private void g ()
    {
        /* implémentation */
    }

    /** Attribut */
    private int m_i;
}
```

FIG. 16 – Une classe équivalente à celle de la Figure 15 en `java`

– Pour déclarer un pointeur d'un certain type, il faut faire suivre le nom du type d'une `*` lors de la déclaration.
– Pour déréférencer un pointeur, on utilise l'opérateur `*` et pour avoir l'adresse d'une variable, on utilise l'opérateur `&`.
– Pour appeler une méthode par un pointeur, on utilise l'opérateur `->`.

12.2 Arithmétique des pointeurs

Les additions et les soustractions d'entiers sont permises sur les pointeurs. Ajouter x à un pointeur revient à ajouter au pointeur x fois le nombre d'octets occupés par le type pointé.

L'opérateur `sizeof` permet de connaître le nombre d'octets occupés par une instance d'un certain type.

Un tableau et un pointeur sont équivalents.

12.3 Gestion de la mémoire

Les opérateurs `new` et `delete` permettent de gérer vous mêmes votre mémoire. Attention aux fuites et aux pointeurs foux.

13 Template

Les type générique permette une très grande abstraction sur les types. Ce sont des genres d'interfaces implicites.

13.1 Utilisation

14 Création d'un programme

Comme en `java`, le point d'entré du programme est une fonction nommé `main`. Cependant, le `main` en `c++` ne doit pas être dans une classe et doit avoir la signature suivante : `int main (int argc, char* argv[])`

Voyez la Figure 18 pour un exemple.

15 Utilisation d'un compilateur linux

Le compilateur que vous utiliserez fait partie de la suite gnu et se nomme `g++`. Vous pourrez l'appeler de la façon suivante : `g++ -g -Wall -o nom_programme fichier.cc`. Lancez `man g++` pour plus de détails.

```
#include <iostream>

using namespace std;

class Chainon
{
public :
    Chainon (int v, Chainon* s = 0)
    {
        valeur = v;
        suivant = s;
    }

    int valeur;
    Chainon* suivant;
};

int main() {
    Chainon * courant, * tete;
    int i;

    tete = 0;

    for(i=1;i<=10;i++) {
        courant = new Chainon (i);
        courant->suivant  = tete;
        tete = courant;
    }

    for (courant = tete; courant != 0; courant = courant->suivant)
    {
        cout << courant->valeur << " ";
    }

    cout << endl;

    return 0;
}
```

FIG. 17 – Pointeurs en c++

```
// faire les include

using namespace std;

// déclarer les fonctions qui seront utilisés dans le main

...

/**
 * La fonction principale.
 *
 * @param argc Le nombre de paramètre sur la ligne de
 *             commande.
 * @param argv Un tableau de chaîne de caractères
 *             contenant les arguments
 *
 * @return 0 en cas de succès.
 */
int main (int argc, char* argv[])
{
    // Faire le traitement

    return 0; // tout s'est bien passé
}
```

FIG. 18 – Exemple d'un programme c++