Faculté des Sciences et de la Technologie

Programmation Orienté Objet en Java

11 mars 2024

Enseignant : HAMDANI M

Filière : Informatique

Spécialité : Informtique
Semestre : S 4

Plan

Introduction

Le langage C++ a été conçu à partir de 1982 par Bjarne Stroustrup (AT&T Bell Laboratories), dès 1982, comme une extension du langage C, lui-même créé dès 1972 par Denis Ritchie, formalisé par Kerninghan et Ritchie en 1978. L'objectif principal de B. Stroustrup était d'ajouter des classes au langage C et donc, en quelque sorte, de « greffer » sur un langage de programmation procédurale classique des possibilités de « programmation orientée objet » (en abrégé P.O.O.).

Créateurs

Dennis Ritchie
1941 - 2011

Brian Wilson Kernighan
(1942)

Bjarne Stroustrup (1950)

Problématique de la programmation

- l'exactitude : aptitude d'un logiciel à fournir les résultats voulus, dans des conditions normales d'utilisation (par exemple, données correspondant aux spécifications) ;

Problématique de la programmation

- l'exactitude : aptitude d'un logiciel à fournir les résultats voulus, dans des conditions normales d'utilisation (par exemple, données correspondant aux spécifications) ;
- la robustesse : aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation ;

Problématique de la programmation

- l'exactitude : aptitude d'un logiciel à fournir les résultats voulus, dans des conditions normales d'utilisation (par exemple, données correspondant aux spécifications) ;
- la robustesse : aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation ;
- l'extensibilité : facilité avec laquelle un programme pourra être adapté pour satisfaire à une évolution des spécifications ;

Problématique de la programmation

- l'exactitude : aptitude d'un logiciel à fournir les résultats voulus, dans des conditions normales d'utilisation (par exemple, données correspondant aux spécifications) ;
- la robustesse : aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation ;
- l'extensibilité : facilité avec laquelle un programme pourra être adapté pour satisfaire à une évolution des spécifications ;
- la réutilisabilité : possibilité d'utiliser certaines parties (modules) du logiciel pour résoudre un autre problème ;

Problématique de la programmation

- l'exactitude : aptitude d'un logiciel à fournir les résultats voulus, dans des conditions normales d'utilisation (par exemple, données correspondant aux spécifications) ;
- la robustesse : aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation ;
- l'extensibilité : facilité avec laquelle un programme pourra être adapté pour satisfaire à une évolution des spécifications ;
- la réutilisabilité : possibilité d'utiliser certaines parties (modules) du logiciel pour résoudre un autre problème ;
- la portabilité : facilité avec laquelle on peut exploiter un même logiciel dans différentes implémentations ;

Problématique de la programmation

- l'exactitude : aptitude d'un logiciel à fournir les résultats voulus, dans des conditions normales d'utilisation (par exemple, données correspondant aux spécifications) ;
- la robustesse : aptitude à bien réagir lorsque l'on s'écarte des conditions normales d'utilisation ;
- l'extensibilité : facilité avec laquelle un programme pourra être adapté pour satisfaire à une évolution des spécifications ;
- la réutilisabilité : possibilité d'utiliser certaines parties (modules) du logiciel pour résoudre un autre problème ;
- la portabilité : facilité avec laquelle on peut exploiter un même logiciel dans différentes implémentations ;
- l'efficience : temps d'exécution, taille mémoire.

Programmation structurée

Équation de Wirth :
Programmes = algorithmes + structures de données

Objet

Une association des données et des procédures (Méthodes)

Méthodes + Données = Objet

Encapsulation

- Il n'est pas possible d'agir directement sur les données d'un objet ;
- Il est nécessaire de passer par l'intermédiaire de ses méthodes, qui jouent ainsi le rôle d'interface obligatoire.

Encapsulation

- Il n'est pas possible d'agir directement sur les données d'un objet ;
- Il est nécessaire de passer par l'intermédiaire de ses méthodes, qui jouent ainsi le rôle d'interface obligatoire.
- L'encapsulation facilite considérablement la maintenance : une modification éventuelle de la structure des données d'un objet n'a d'incidence que sur l'objet lui-même
- L'encapsulation des données facilite grandement la réutilisation d'un objet.

Classe

- Généralisation de la notion de type que l'on rencontre dans les langages classiques. ;

Classe

- Généralisation de la notion de type que l'on rencontre dans les langages classiques. ;
- Description d'un ensemble d'objets ayant une structure de données commune2 et disposant des mêmes méthodes

Classe

- Généralisation de la notion de type que l'on rencontre dans les langages classiques. ;
- Description d'un ensemble d'objets ayant une structure de données commune2 et disposant des mêmes méthodes
- Les objets apparaissent alors comme des variables d'un tel type classe -> on dit aussi qu'un objet est une « instance » de sa classe.

Héritage (inheritance)

- Permet de définir une nouvelle classe à partir d'une classe existante, à laquelle on ajoute de nouvelles données et de nouvelles méthodes

Héritage (inheritance)

- Permet de définir une nouvelle classe à partir d'une classe existante, à laquelle on ajoute de nouvelles données et de nouvelles méthodes
- Facilite largement la réutilisation de produits existants

Héritage (inheritance)

- Permet de définir une nouvelle classe à partir d'une classe existante, à laquelle on ajoute de nouvelles données et de nouvelles méthodes
- Facilite largement la réutilisation de produits existants
- La classe **C** peut hériter de **B**, qui elle-même hérite de **A**.

Polymorphisme

- polymorphisme vient du grec et signifie qui peut prendre plusieurs formes

Polymorphisme

- polymorphisme vient du grec et signifie qui peut prendre plusieurs formes
- Une classe dérivée peut « redéfinir » (c'est-à-dire modifier) certaines des méthodes héritées de sa classe de base

Polymorphisme

- polymorphisme vient du grec et signifie qui peut prendre plusieurs formes
- Une classe dérivée peut « redéfinir » (c'est-à-dire modifier) certaines des méthodes héritées de sa classe de base
- On utilise chaque objet comme s'il était de cette classe de base, mais son comportement effectif dépend de sa classe effective.

La surcharge (overloading)

- La surcharge (surdéfinition) est une possibilité offerte par certains langages de programmation qui permet de choisir entre différentes implémentations d'une même fonction ou méthode selon le nombre et le type des arguments fournis.

La surcharge (overloading)

- La surcharge (surdéfinition) est une possibilité offerte par certains langages de programmation qui permet de choisir entre différentes implémentations d'une même fonction ou méthode selon le nombre et le type des arguments fournis.
- (employer plusieurs fonctions de même nom)

Déclarations

```
int i ;
float x ;
float racx, y ;
const int NFOIS = 5 ;
```

Pour écrire des informations : utiliser le flot ***cout***

```
cout << "Bonjour\n" ;
cout << "Je vais vous calculer " << NFOIS << "
      racines carrées \n" ;
```

Pour lire des informations : utiliser le flot ***cin***

```
cin » x ;
```

L'instruction *if*

```
if (expression)
     instruction 1
else
    instruction 2
```

L'instruction *if* - *exemple*

```
1   #include <iostream>
2   using namespace std ;
3   main()
4   {   const double TAUX_TVA = 19.6 ;
5     double ht, ttc, net, tauxr, remise ;
6     cout << "donnez le prix hors taxes : " ;
7     cin >> ht ;
8     ttc = ht * ( 1. + TAUX_TVA/100.) ;
9     if ( ttc < 1000.)              tauxr = 0 ;
10    else if ( ttc < 2000 )         tauxr = 1. ;
11         else if ( ttc < 5000 )    tauxr = 3. ;
12              else                 tauxr = 5. ;
13                   remise = ttc * tauxr / 100. ;
14    net = ttc - remise ;
15    cout << "prix ttc = " << ttc << "\n" ;
16    cout << "remise = " << remise << "\n" ;
17    cout << "net à payer = " << net << "\n" ;
18  }
```

L'instruction *switch*

```
1     #include <iostream>
2   using namespace std ;
3   main()
4   { int n ;
5     cout << "donnez un entier : " ;
6     cin >> n ;
7     switch (n)
8     { case 0 : cout << "nul\n" ;
9       break ;
10      case 1 : cout << "un\n" ;
11      break ;
12      case 2 : cout << "deux\n" ;
13      break ;
14      default : cout << "grand\n" ;
15    }
16    cout << "au revoir\n" ;
17  }
```

L'instruction *do ... while*

```
1   #include <iostream>
2   using namespace std ;
3   main()
4   { int n ;
5     do
6     { cout << "donnez un nb > 0 : " ;
7       cin >> n ;
8       cout << "vous avez fourni : " << n << "\n" ;
9     }
10    while (n >= 0) ;
11    cout << "reponse correcte" ;
12  }
```

L'instruction *while*

```
1   #include <iostream>
2   using namespace std;
3   int main () {
4     int a = 10;
5     while( a < 20 ) {
6       cout << "value of a: " << a << endl;
7       a++;
8     }
9      return 0;
10  }
```

L'instruction *for*

```
1   #include <iostream>
2   using namespace std ;
3   main()
4   {
5     int i ;
6     for ( i=1 ; i<=5 ; i++ ){
7       cout << "bonjour " ;
8       cout << i << " fois\n" ;
9     }
10  }
```

Les instructions de branchement inconditionnel : *break*, *continue* et *goto*

L'instruction break : sert à interrompre le déroulement de la boucle, en passant à l'instruction qui suit cette boucle

L'**instruction continue** : permet de passer prématurément au tour de boucle suivant.

L'instruction goto : permet classiquement le branchement en un emplacement quelconque du programme.

les fonctions : 1er - exemple

```
1   #include <iostream>
2   using namespace std ;
3   int fact (int n) {
4     int f = 1;
5     for (int i=1; i<=n; i++)
6       f=f*i;
7     return(f);
8     }
9
10   void main (String[] args) {
11     System.out.print(fact(6));
12    System.out.print("\n");
13  }
```

Arguments muets et arguments effectifs

- Les noms des arguments figurant dans l'en-tête de la fonction se nomment des « arguments muets », ou encore « arguments formels » ou « paramètres formels » (de l'anglais : formal parameter).

- Les arguments fournis lors de l'utilisation (l'appel) de la fonction se nomment des « arguments effectifs » (ou encore « paramètres effectifs »).

Transmission des arguments par valeur

```
1   #include <iostream>
2   using namespace std ;
3
4   main(){
5    void echange (int a, int b) ;
6    int n=10, p=20 ;
7    cout << "avant appel : " << n << " " << p << "\n" ;
8    echange (n, p) ;
9    cout << "apres appel : " << n << " " << p << "\n" ;
10  }
11
12  void echange (int a, int b){
13   int c ;
14   cout << "début echange : "<< a << " " << b << "\n" ;
15   c = a ;
16   a = b ;
17   b = c ;
18   cout << "fin echange : " << a << " " << b << "\n" ;
19  }
```

Résultats : ?

Transmission des arguments par référence

```
1   #include <iostream>
2   using namespace std ;
3
4   main(){
5     void echange (int & , int & ) ;
6     int n=10, p=20 ;
7     cout << "avant appel : " << n << " " << p << "\n" ;
8     echange (n, p) ;
9     cout << "apres appel : " << n << " " << p << "\n" ;
10  }
11
12  void echange (int & a, int & b){
13    int c ;
14    cout << "début echange : "<< a << " " << b << "\n" ;
15    c = a ;
16    a = b ;
17    b = c ;
18    cout << "fin echange : " << a << " " << b << "\n" ;
19  }
```

Résultats : ?

Transmission des arguments

```
Transmission des arguments par valeur :
 avant appel : 10 20
 début echange : 10 20
 fin echange : 20 10
 apres appel : 10 20

Transmission des arguments par référence :
 avant appel : 10 20
 début echange : 10 20
 fin echange : 20 10
 après appel : 20 10
```

Récursivité

- récursivité directe : une fonction comporte, dans sa définition, au moins un appel à elle même ;

- récursivité croisée : l'appel d'une fonction entraîne celui d'une autre fonction qui, à son tour, appelle la fonction initiale (le cycle pouvant d'ailleurs faire intervenir plus de deux fonctions).

```
long fac (int n) {
  if (n>1)
    return (fac(n-1)*n) ;
  else
    return(1) ;
}
```

tableaux

```
1   #include <iostream>
2   using namespace std ;
3   const int N = 2;
4   const int M = 3;
5   int main(){
6     int i, j;
7     int t[N][M];
8     for(i=0; i<N; i++)
9        for(j=0; j<M; j++){
10          cout<<"Tapez t["<< i <<"]["<< j <<"] :";
11        cin >> t[i][j];
12       }
13
14    cout<<"Voici le tableau :"<<endl;
15    for(i=0; i<N; i++){
16      for(j=0; j<M; j++) cout<< t[i][j] <<" ";
17      cout<<endl;
18    }
19    return 0;
20  }
21
```

Généralité

La règle générale pour créer un fichier est la suivante :

- Ouvrir le fichier en écriture.
- Écrire des données dans le fichier.
- Fermer le fichier.

Pour lire des données écrites dans un fichier :

- Ouvrirle fichier en lecture.
- Lire les données en provenance du fichier.
- Ferme le fichier.

Il existe principalement 2 bibliothèques standard pour écrire des fichiers :

- ***cstdio*** qui provient en fait du langage C.
- ***fstream*** qui est typiquement C++.

Écriture dans un fichier

```
1  #include <iostream>
2  #include <fstream>
3  #include <string>
4  using namespace std;
5  int main(){
6    string const nomFichier("C:/Nanoc/scores.txt");
7    ofstream monFlux(nomFichier.c_str());
8
9    if(monFlux){
10     monFlux << "Bonjour, je suis une phrase écrite dans un fichier." <<
                  endl;
11     monFlux << 42.1337 << endl;
12     int age(23);
13     monFlux << "J'ai " << age << " ans." << endl;
14   }
15   else{
16     cout << "ERREUR: Impossible d'ouvrir le fichier." << endl;
17   }
18   return 0;
19 }
```

Questions ?