Langages objets

Opérateurs et expressions
Instructions de contrôle
Débugage

M2 Pro CCI, Informatique
Emmanuel Waller, LRI, Orsay

Les opérateurs et les expressions

- classiques : arithmétiques, relationnels, logiques
- manipulation de bits
- affectation, incrémentation
- règles de priorité
- règles de conversion de type

originalité des notions d'opérateur et d'expression

- en général (hors Java, C, C++) :
  - expressions : formées à l'aide d'opérateurs :
    - ont une valeur
    - ne font rien
  - instructions :
    - font quelque chose
  - n'ont pas de valeur

- Java :
  - i++
    - a une valeur
    - fait quelque chose
  - i = 5;
    - fait quelque chose
    - a une valeur
  - expression; est une instruction (conséquences pour fonctions)

les opérateurs arithmétiques

- présentation des opérateurs
  - binaires : + - * / : opérandes de même type (int, long, float, double), mais conversions implicites permettent aussi byte, char, short
  - unaire : - +
  - % (modulo) : entiers, flottants
  - 7 / 2 vaut 3 : entier
- les priorités relatives des opérateurs
  - règles
  - désambigüer par parenthèses

- comportement en cas d'exception
  - il y a circonstances où un opérateur ne peut fournir un résultat correct
  - entiers : division par zéro : arrêt du programme + message
  - flottants : jamais d'arrêt : Infinity, -Infinity, NaN

les conversions implicites dans les expressions

- ex :
  - int n, p; double x; (n * x) + p
  - n et p convertis par compilateur en double
  - résultat : double
- règles, type char, être soigneux, nombreux cas, surprises, subtilités, ...

les opérateurs relationnels

- < <= > >= == !=
- renvoient un booléen
- == et != s'appliquent à : booléens, objets, tableaux

les opérateurs logiques

- ! : négation
- & : et
- ^ : ou exclusif
- | : ou inclusif
- && : et, avec court-circuit
- || : ou inclusif, avec court-circuit
- ex : if (i<t.length && t[i] == 0) ...

l'opérateur d'affectation usuel : =

- i = 5 est une expression qui :
  - effetcue une action : l'affectation de la valeur 5 à i
  - possède une valeur : celle de i après affectation : 5
- c = b + 3 : expression
- à gauche de = : référence à un emplacement dont on peut modifier la valeur (variables, etc.)
- possible : i = j = 5 :
  - évalue j = 5, qui vaut 5, et l'affecte à i
  - valeur finale de toute l'expression : 5
- subtilités de conversion cause affectation

les opérateurs d'incrémentation et de décrémentation

- l'opérateur ++ : but : remplacer i = i + 1
- ++i
  - incrémente i de 1
  - valeur : i après incrémentation
- i++ : idem, valeur : i avant incrémentation
- i++; équivalent ++i;
- -- décrémentation
- tous types numériques

les opérateurs d'affectation élargie

- i += k équivalent à i = i + k
- += -= *= /= %= ^= &= (et aussi manipulations de bits : <<= >>= <<<=)
- subtilités conversions

l'opérateur de cast

- le programmeur peut forcer la conversion d'une expression dans un autre type de son choix avec cast
- ex : int n, p; (double) (n/p) : valeur :
  1. calcul de n/p : entier
  2. conversiondu résultat en double

les opérateurs de manipulation de bits

- travailler sur le motif binaire (octets bit à bit) d'une valeur
- bit à bit : &, |, ^
- décalage : <<, >>, >>>, ~

l'opérateur conditionnel

- if (a>b)

  max = a;

  else

  max = b;
- max = si (a>b) alors a sinon b
- max = a>b ? a : b

opérateurs et expressions : récapitulatif

- Originalité des notions d'opérateur et d'expression
- Les opérateurs arithmétiques
- Les conversion implicites dans les expressions
- Les opérateurs relationnels
- Les opérateurs logiques
- L'opérateur d'affectation usuel
- Les opérateurs d'incrémentation et de décrémentation : ex : i++, i--
- Les opérateurs d'affectation élargie : ex : +=
- L'opérateur de cast
- Les opérateurs de manipulation de bits
- L'opérateur conditionnel
- (Delannoy chapitre 4)

Les instructions de contrôle de Java

- séquencement
- bloc
- if
- switch
- do ... while
- while
- for
- break et continue

Le séquencement

- le corps d'une fonction est une séquence d'instructions
- elles sont exécutées l'une après l'autre dans cet ordre (sauf cas particuliers présentés plus loin : break, continue, exceptions)

le bloc d'instructions

- suite d'instructions (simple, de contrôle, bloc) placées entre accolades { et }
- ex : vu dans chap. 1
- ex inutiles :
  - { }
  - { i = 1; }
  - { ; } // rappel chap. 2 : existe instruction vide : ;
- remarque : inutile ajouter ; après { . . . }

L'instruction if

- syntaxe :

```
if (condition)
    instruction1
[ else
    instruction2 ]
```

- condition :
  - booléenne quelconque
  - parenthèses obligatoire toute condition Java
- instruction1 et instruction2 quelconques : simple, de contrôle, bloc
- [. . .] : facultatif

imbrication des instructions if

- ex : if (a) if (b) c else d
- else correspond-il à if (a) ou if (b) ?
- différent : ex : a faux : d ou rien ?
- un else se rapporte toujours au dernier if rencontré auquel un else n'a pas encore été attribué
- ex : if (a) {if (b) c else d}

  ne rien faire si a faux

exemple

- 2 paramètres, afficher expression sur le 2ème en fonction différentes valeurs du premier
- ci-joint
- démonstration

L'instruction switch

- ex : à partir d'un entier n, afficher sa valeur en français, et « grand » s'il est trop grand (en gros)

```
int n;
n = ... lecture clavier ...
switch (n) {
   case 0 : System.out.println("zéro");
            break;
   case 1 : System.out.println("un");
            break;
   case 3 : System.out.println("trois");
            break;
   default : System.out.Println("grand");
}
System.out.println("Au revoir");
```

1. évaluer expression switch (expr) : vaut i
2. rechercher dans bloc une étiquette case x, x expression constante int, où x est cette valeur
3. si existe se brancher à cette instruction

   sinon passer à l'instruction qui suit le bloc switch

- ex : n = 0 :

  zéro

  Au revoir
- sans les break : n = 0 :

  zéro

  un

  trois

  Au revoir

- l'étiquette default : on s'y branche si aucun étiquette ne correspond

```
switch (n) {
   case 0 : System.out.println("zéro");
            break;
   case 1 : System.out.println("un");
            break;
   default : System.out.Println("grand");
}
System.out.println("Au revoir");
```

- ex : n = 3 : grand

```
switch (n) {
   case 0 : System.out.println("zéro");
            break;
   case 1 :
   case 2 : System.out.println("petit");
   case 3 :
   case 4 :
   case 5 : System.out.println("moyen");
            break;
   default : System.out.Println("grand");
}
System.out.println("Au revoir");
```

- ex : n = 1

  petit

  moyen

  Au revoir
- possible :
  - plusieurs instructions par étiquette
  - étiquettes sans instruction

exemple

- 2 paramètres, afficher expression sur le 2ème en fonction différentes valeurs du premier
- ci-joint
- démonstration

l'instruction do . . . while

- ex : à partir d'un entier, lui retirer 3 et afficher ce qu'il reste en recommençant jusqu'à arriver à zéro

```
int n = 29;
do {
    n = -3;
    System.out.println("il en reste" + n);
} while (n >= 2);
```

- répète instruction (ici bloc) tant que condition vraie

exemple

- Ci-joint
- Démonstration

- condition testée que après que tout le bloc fini (donc corps au moins une fois)
- do instruction while (condition)
- n est la « variable de boucle » : elle contrôle les passages

exemples

- do ; while (...); // infinie ? action ?
- do { } while (...); // idem ?
- do { } while (true); // idem ?
- do instruction while (true);
  // utile si sortie par break

l'instruction while

- ex : payer un café à 29 centimes avec des pièces de 3 centimes

```
int n = 0;
while (n < 29) {
    System.out.println("il manque encore" + (29 - n));
    n += 3;    // ici on paye
}
```

- répète instruction (ici bloc) tant que condition vraie

exemple

- Ci-joint
- Démonstration

- condition examinée avant corps boucle (donc possible corps jamais)
- while (condition) instruction

l'instruction for

- vue dans chap. 2
- for (i=1, j=3; i<5; i++, j+=i) { ... }

  suite d'expressions séparés par des virgules
- for (int i=1, j=3; . . .) { ... }

  ou une déclaration : int une seule fois, puis variables, éventuellement initialisées (sinon ici inutile), séparées par des virgules
- portée de i et j : corsp du for, inconnues après

- for ([initialisation]; [condition]; instruction) {

  . . .

  }
- condition absente considérée comme vraie
- initialisation : choix exclusif : déclaration ou liste d'expressions (virgule n'est pas un opérateur, cf C, C++)

  impossible : for (int i=1, double x=0; ...) { ... }
- attention compteur non entier (erreur d'arrondi) :

  for (double x=0.; x!=1.0; x+=1) { ... } : infinie

- exemples :

  for (; ;) ; // infinie ? action ?

  for (; ;) { } // idem ?

  for (; ;) instruction // idem ?
- remarque
  - for est une boucle conditionnelle
  - pas vraie boucle avec compteur
  - for (i=0; i<5; i++) { ... i-- ... } : possible, déconseillé

Boucles imbriquées : exemple 1

- Afficher 5 lignes, contenant chacune 0 1 2 3 4
- Principe : transparent suivant
- Ci-joint
- Démonstration

- pour i de 0 à 4

  afficher la ligne
- afficher la ligne =

  pour j de 0 à 4

  print j

  println
- pour i de 0 à 4

  pour j de 0 à 4

  print j

  println
- Noter le bloc dans la boucle extérieure

Boucles imbriquées : exemple 2

- Afficher 5 lignes, la première contenant 0, la deuxième 0 1, etc. la dernière 0 1 2 3 4
- Ci-joint
- Démonstration

Les instructions de branchement inconditionnel break et continue

break

- déjà vu : dans switch
- possible dans les trois boucles :
  - interrompt le déroulement de la boucle en passant à l'instruction suivant la boucle
  - utile que si break dépend d'un if (sinon sortie dès premier tour)
  - si boucles imbriquées, break sort uniquement de la plus interne
- impossible hors boucle ou switch
- sortir de plusieurs boucles : break avec étiquette

continue

- permet de passer prématurément au tour de boucle suivant (la fin du tour en cours n'est pas faite)
- branchement avant les « incrémentations »
- si boucles imbriquées : concerne que interne
- sortie plusieurs niveaux : continue avec étiquette

```
for (int i=1; i<=4; i++) {
   System.out.println("début du tour " + i);
   if (i<3) continue;
   System.out.println("fin du tour " + i);
}
System.out.println("après la boucle");
```

```
début du tour 1
début du tour 2
début du tour 3
fin du tour 3
début du tour 4
fin du tour 4
après la boucle
```

les instructions de contrôle : récapitulatif

- séquencement
- bloc
- if
- switch
- do ... while
- while
- for
- break et continue
- (Delannoy chapitre 5)

remarque

- syntaxe Java identique à C et C++ (sauf détail syntaxe programme principal et déclaration fonction)
- autrement dit : « Java sans objets c'est simplement C » (Java conçu pour)
- Toujours pas d'objets

Exemple : tirages aléatoires

- Bibliothèque Math
- Indépendant du chapitre « instructions de contrôle »
- Tirer un nombre réel aléatoire dans [0,1[
  (vérifier si bornes ouvert ou fermé)
- Ci-joint
- Démonstration

débugage (et méthodologie de développement)

débugage (et méthodologie de développement)

- méthodologie de développement
- notion d'erreur
- erreurs lors de la compilation
- erreurs lors de l'exécution
- exemple

« méthodologie de développement »

- résoudre le problème sur un exemple à la main
- écrire algorithme/modules en français sur papier
  - données
  - traitements
- écrire le code Java
  - faire tourner totalement un fragment du problème
  - l'étendre
  - rem : exécuter programme chaque fois que ajouté 5 lignes
- tester et débuguer
- mieux : cahier des charges, spécification, tests, codage

notion d'erreur

- deux catégories :
  - syntaxe : détectées lors de la compilation
  - sémantique (logique) : constatées pendant l'exécution
- Un informaticien (même débutant) doit systématiquement :
  - Les lire
  - Les comprendre en totalité
- la fenêtre doit être assez grande pour voir la totalité des messages

erreurs lors compilation

- fichier source peut présenter erreurs de syntaxe
- compilateur :
  - les détecte
  - affiche pour chaque erreur message indiquant ce qu'il a compris
  - ne génère pas d'exécutable

causes erreurs compilation

- environnement : fichier pas trouvé par compilateur
  - fichier pas dans répertoire, non sauvegardé, erreur d'extension, de casse, d'orthographe (coquille), etc.
- syntaxe
  - résoudre une seule erreur, la première, puis recompiler (elle peut avoir créé les autres)
  - message d'erreur : numéro de ligne du fichier, la ligne, ce que le compilateur comprend, ce qu'il attendait

erreurs pendant l'exécution

- exécution d'un programme compilé peut conduire à des « opérations » qui n'ont pas de sens en Java : erreurs sémantiques (logiques)
  - ex : division par zéro, indice tableau hors bornes
- la JVM :
  - interrompt l'exécution
  - affiche le contexte du programme au moment de l'erreur

causes erreurs exécution

- environnement : mêmes problèmes que compilation, pas le bon nom de classe
- sémantique (logique) : détailler le message
  - une ligne par fonction en cours au moment de l'erreur (ex : f(g(h(x))))
  - valeur des paramètres et ligne du fichier source
  - se lit de bas en haut
  - etc.

Exemple

- Expliquer ce qui a mené aux messages d'erreur (indépendants) suivants

```
Exception in thread "main"
        java.lang.ArrayIndexOutOfBoundsException: 6
    at Liste.creer(Listes.java:22)
    at Listes.main(Listes.java:56)
```

```
Listes.java:20: cannot resolve symbol
symbol  : variable args
location: class Liste
    tmp.suivant = new Liste(Integer.parseInt(args[i]));
                                             ^
```

« méthodologie pour débuguer »

- bugs sémantiques (à l'exécution)
- localiser le bug : en affichant toutes les variables

délégués ?