CSI 1502
Principes fondamentaux de conception de logiciels

Chapitre 3:
Instructions de base de Java

Objectifs d'apprentissage: Instructions de base de Java

- Comprendre les conceptes de "flot de contrôle" dans une méthode
- Structures de sélection: *if*, *if else* et *switch*
- Opérateurs
  - Booléens: AND, OR, NOT
  - Autres: incrément ++, décrément --, assignation +=, et conditionnel ?
- Structures de répétition: *while*, *do* et *for*
- Comprendre les étapes de développement importantes

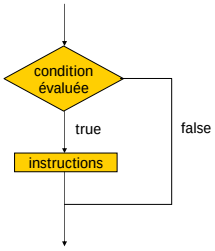
Qu'est-ce le "flot de contrôle"?

- Certaines structures de contrôle modifient le flot linéaire de contrôle, qui nous permet de:
  - Décider si on devrait exécuter des instructions particulières, ou
  - Exécuter une instruction plusieurs fois, de façon répétitive
- Ces décisions sont basées sur une expression booléenne (appelée condition) qui évalue à la valeur *true* ou *false*
- **L'ordre d'exécution des instructions est appelé le "flot de contrôle"**

Structure de sélection de type 1: La structure **if**

```
if ( condition )
    instructions ;
```

- *if* est un terme réservé dans Java
- la *condition* doit être une expression booléenne qui évalue à *true* ou *false*
- si la *condition* est vraie, les *instructions* sont exécutées
- si la *condition* est fausse, les *instructions* sont ignorées

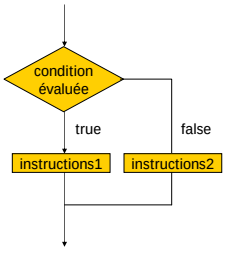


La structure **if**: Un exemple Age.java

```
import cs1.Keyboard;

public class Age
{
// Lecture de votre age composé au clavier et imprime une remarque

   public static void main (String[] args)
   {
        final int MINOR = 21;

        System.out.println("Entrez votre age: ");
        int age = Keyboard.readInt();

        if (age < MINOR)
               System.out.println("Youth is a wonderful thing.");

        System.out.println("Age is a state of mind.");
   }
}
```


Structure de sélection de type 2: La structure **if-else**

```
if ( condition )
    instructions1 ;
else
    instructions2;
```



exemple:

```
if (hauteur <= MAX)
   ajustement = 0;
else
   ajustement = MAX - hauteur;
```

- si la *condition* est vraie, les instructions *instructions1* sont exécutées
- si la *condition* est fausse, les instructions *instructions2* sont exécutées
- l'une ou l'autre liste d'instructions sera exécutée, mais pas les deux

Structure **if** imbriquée: MinDeTrois.java

```
import cs1.Keyboard;

public class MinDeTrois
{
// Lecture de 3 entiers composé au clavier et imprime le plus petit

   public static void main (String[] args)
   {
        int num1, num2, num3, min = 0;

        System.out.println("Entrez trois entiers: ");
        num1 = Keyboard.readInt();
        num2 = Keyboard.readInt();
        num3 = Keyboard.readInt();
```


Structure **if** imbriquée: MinDeTrois.java (cont)

```
        if ( num1 < num2 )
                if ( num1 < num3 )
                        min = num1;
                else
                        min = num3;
        else
                if ( num2 < num3 )
                        min = num2;
                else
                        min = num3;

        System.out.println("Valeur minimum: " + min);
    }
}
```

Résultats

```
Entrez trois entiers:
21341  3424  1233
Valeur minimum: 1233
```


Structure de sélection de type 3: La structure **switch**

- la syntaxe générale pour la structure *switch* est:

```
switch ( expression )
{
  case valeur1 : liste-d-instructions1
  case valeur2 : liste-d-instructions2
  case valeur3 : liste-d-instructions3
  case ...
}
```

- *switch* et *case* sont des termes réservés dans Java
- si *expression* correspond à la valeur *valeur2*, le contrôle saute à la *liste d'instructions2*
- commandes importantes: break, default

La commande **switch**: RapportDeNotes.java

```
import cs1.Keyboard;

public class RapportDeNotes
{
// Lecture de note de cours composé au clavier et imprime une
   remarque

   public static void main (String[] args)
   {
        int note, catégorie;

        System.out.println("Entrez une note de cours numérique: ");
        note = Keyboard.readInt();

        catégorie = note/10;

        System.out.println("C'est ");
```


La commande **switch**: RapportDeNotes.java (cont)

```
        switch ( catégorie )
        {
                case 10:
                        System.out.println("excellent.");
                        break;
                case 9:
                        System.out.println("très bien.");
                        break;
                case 8:
                        System.out.println("bien.");
                        break;
                default:
                        System.out.println("une faillite.");
        }
    }
}
```

Résultats

```
Entrez une note de cours numérique:
87
C'est bien.
```


Structure de sélection de type 3: La structure **switch**

- Le résultat de l'*expression* de la structure *switch* doit être de type *int* ou de type *char* (pas de type *boolean*, *float*, *double*, *byte*, *short* ou *long*)
- La condition booléenne implicite dans une structure *switch* est l'égalité- on essaie pour un match entre l'*expression* et une des v*aleurs*
- **Comment implémenter le *switch* d'une autre façon?**

Bloc d'instructions

- On peut grouper plusieurs instructions ensemble dans un bloc
- Un bloc est délimité par des accolades: { ... }
- Toute instruction dans le code Java peut être un bloc d'instructions (i.e. être délimitée par des accolades)
- Par exemple, dans une structure *if-else*, la section *if* ou la section *else* (ou les deux) peut être un bloc d'instructions
- Voir Guessing.java (page 141)

13

Expressions booléennes

- Une condition utilise souvent un opérateur de comparaison de Java, qui retourne un résultat booléen:

==	Égalité
!=	Différence
<	Inférieur
>	Supérieur
<=	Inférieur ou égal
>=	Supérieur ou égal

E.g. (age != 21) (age >= 21) (age == 21)

14

Opérateurs logiques et tables de vérité

- Les expressions booléennes peuvent utiliser les opérateurs logiques suivants:

!	NOT logique
&&	AND logique
||	OR logique

- Ces opérateurs nécessitent des opérandes booléens et produisent un résultat booléen
  - Le NOT logique est un opérateur unaire (à un seul opérande)
    e.g. !trouvé
  - Le AND logique et le OR logique sont des opérateurs binaires (qui utilisent deux opérandes)
    e.g. (Age != 60) && !trouvé

15

Tables de vérité

- Une table de vérité démontre les combinaisons vrai/faux des termes
- Puisque && et || ont deux opérandes, il y a quatre combinaisons possibles pour les conditions a et b

a	b	a && b	a || b
vrai	vrai	vrai	vrai
vrai	faux	faux	vrai
faux	vrai	faux	vrai
faux	faux	faux	faux

- Les tables de vérité sont utiles pour le débogage!

16

Opérateurs logiques

- Les conditions peuvent utiliser les opérateurs logiques pour construire des expressions complexes

```
if (total < MAX+5 && !trouvé)
   System.out.println ("Traitement en marche...");
```

- Les opérateurs logiques ont des relations de précédence entre eux et avec autres opérateurs
  - Tous les opérateurs logiques ont une priorité inférieure par rapport aux opérateurs arithmétiques et de comparaison
  - L'opérateur logique NOT a une priorité supérieure par rapport au AND logique et au OR logique

17

Attention: Les opérateurs "court-circuit"

- Le traitement du AND logique et du OR logique peut subir un "court-circuit"
- Si l'opérande gauche d'une expression de condition est suffisant pour évaluer le résultat, l'opérande droit ne sera pas évalué

```
if (compteur != 0 && total/compteur > MAX)
      System.out.println ("Vérification...");
```

- **Ce type de traitement est possiblement dangereux: Pourquoi?**

18

La comparaison de caractères

- Il est possible d'utiliser les opérateurs de comparaison sur les données de caractères
- Les résultats sont basés sur le jeu de caractères Unicode
- La condition suivante est vraie parce que le caractère '+' se trouve avant le caractère 'J' dans le jeu de caractères Unicode

```
if ('+' < 'J')
   System.out.println ("+ est plus petit que J");
```

- Les lettres majuscules (A-Z) suivi par les lettres minuscules (a-z) sont inclut en ordre alphabétique dans le jeu de caractères Unicode

19

La comparaison de chaînes de caractères

- Une chaîne de caractères est un objet en Java
  - On ne peut pas utiliser les opérateurs de comparaison si on veut comparer des chaînes de caractères
- La méthode *equals* peut être utilisé pour déterminer si deux chaînes de caractères contiennent les mêmes caractères dans le même ordre

  e.g. `( nom1.equals (nom2) )`
- La classe *String* contient une méthode *compareTo* qui peut être utilisé pour déterminer si une chaîne de caractère précède une autre

  e.g. `int résultat = nom1.compareTo(nom2);`

20

La comparaison de chaînes de caractères: Ordre lexicographique

- La comparaison de caractères et de chaînes de caractères basée sur un jeu de caractères est appelée l'ordre lexicographique
- Ce n'est pas strictement alphabétique lorsqu'il y a un mélange de lettres majuscules et minuscules
  - Par exemple, la chaîne de caractères "Géant" précède la chaîne "fantôme" car les lettres majuscules précèdent les lettres minuscules dans le jeu de caractères Unicode
- Aussi, les chaînes plus courtes précèdent les chaînes plus longues de même préfixe (lexicographique)
  - Par exemple, "fille" précède "fillette"

21

Attention: La comparaison de valeur float

- On devrait rarement utiliser l'opérateur d'égalité (==) pour la comparaison de données de type float, plutôt il est mieux de déterminer si les floats sont "assez proches"
- Pour déterminer l'égalité de deux floats, il est suggéré d'essayer la technique suivante:

```
if (Math.abs(f1 - f2) < 0.00001)
   System.out.println ("À peu près égal.");
```

22

Autres opérateurs de Java

- Opérateurs d'incrémentation (++) et de décrémentation (-)
- Opérateurs d'assignation (+=)
- Opérateurs conditionnels (?)

`compteur++;` est équivalent à

`compteur = compteur + 1;`

23

Incrémentation et décrémentation

- Les opérateurs d'incrémentation et de décrémentation peuvent être appliqué avant l'opérande (forme préfixe) ou après l'opérande (forme postfixe)
- Utilisé seul dans une expression, la forme préfixe et la forme postfixe sont équivalents

`compteur++;`

est équivalent à

`++compteur;`

24

Incrémentation et décrémentation

- Utilisé dans une expression plus complexe, la forme préfixe et la forme postfixe ont différents effets
- Dans les deux cas (pré ou post), la variable est incrémentée ou décrémenté
- Mais la valeur utilisée dans l'expression complexe dépend sur la forme utilisé:

Expression	Opération	Valeur utilisé dans l'expression
compteur++	ajoute 1	ancienne valeur
++compteur	ajoute 1	nouvelle valeur
compteur--	soustrait 1	ancienne valeur
--compteur	soustrait 1	nouvelle valeur

25

Incrémentation et décrémentation

- Si compteur contient 45, alors l'instruction

```
total = compteur++;
```

assigne 45 à **`total`** et 46 à **`compteur`**

- Si compteur contient 45, alors l'instruction

```
total = ++compteur;
```

assigne 46 à **`total`** et à **`compteur`**

26

Opérateurs d'assignation

- Souvent on effectue une opération sur une variable, et ensuite le résultat est stocké dans cette même variable
- Java offre des opérateurs d'assignation qui simplifie le processus
- Par exemple, l'instruction

```
num += compteur;
```

est équivalent à

```
num = num + compteur;
```

27

Opérateurs d'assignation

- Il y a plusieurs opérateurs d'assignation, y compris:

Opérateur	Exemple	Équivalent à
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

28

Opérateurs d'assignation

- Le côté droit d'une opération d'assignation peut être une expression complexe
- Le côté droit au complet est évalué en premier, ensuite le résultat est combiné avec la variable originale
- Alors

```
résultat /= (total - MIN) % num;
```

- est équivalent à

```
résultat = résultat / ((total - MIN) % num);
```

29

Opérateurs d'assignation

- Le comportement de certains opérateurs d'assignation dépend sur le type des opérandes
- Si les opérandes de l'opérateur += sont des chaînes de caractères, l'opérateur d'assignation effectue la concaténation
- Le comportement de l'opérateur += est toujours correspond au comportement de l'opérateur "régulier" +

30

L'opérateur conditionnel

- Java comprend un opérateur conditionnel qui évalue une condition booléenne pour déterminer laquelle des deux expressions sera évalué
- Le résultat de l'expression choisie devient le résultat de l'opérateur conditionnel
- La syntaxe est:

```
condition ? expression1 : expression2
```

- Si la *condition* est vraie, *expression1* est évaluée; si la *condition* est fausse, *expression2* est évaluée

31

L'opérateur conditionnel

- L'opérateur conditionnel est semblable à une structure *if-else*, avec l'exception qu'il forme une expression qui retourne une valeur
- Par exemple:

```
plusgrand = ((num1 > num2) ? num1 : num2);
```

- Si *num1* est plus grand que *num2*, alors *num1* est assigné à la variable *plusgrand*; autrement, *num2* est assigné à *plusgrand*
- L'opérateur conditionnel est ternaire car il nécessite trois opérandes

32

L'opérateur conditionnel

- Un autre exemple:

```
System.out.println ("Votre monnaie est "
+ compteur + ((compteur == 1) ? "pièce de
dix cents" : "pièces de dix cents"));
```

- Si *compteur* est égal à 1, le text "pièce de dix cents" est imprimé
- Si *compteur* est autre que 1, le text "pièces de dix cents" est imprimé

33

Structures de répétition (boucles)

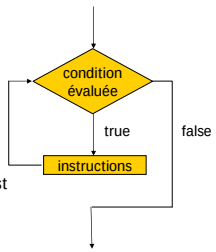
- Java comprend trois structures de répétition:
  - la boucle *while*
  - la boucle *do*
  - la boucle *for*
- Le programmeur devrait choisir la boucle qui correspond mieux à la situation

34

Répétition: La structure *while*

while (condition)
instructions ;

- Si la *condition* est vraie, les *instructions* sont exécutées. Ensuite la *condition* est évaluée encore.
- Les *instructions* sont exécutées de façon itérative jusqu'à ce que la *condition* est fausse
- Note: si la condition d'une boucle while est fausse du départ, les instructions ne sont jamais exécutées
- Le corps de la boucle while sera exécuté zéro ou plus de fois



35

La structure *while* : Moyenne.java

```
import java.text.DecimalFormat;
import cs1.Keyboard;

public class Moyenne
{
   // Calcule la moyenne d'un groupe de valeurs
   public static void main (String[] args)
   {
        int somme = 0, valeur, compteur = 0;
        double moyenne;

        DecimalFormat fmt = new DecimalFormat("0.###");

        System.out.println("Entrez un entier (0 pour sortir)");
        valeur = Keyboard.readInt();
```

36

La structure *while* : Moyenne.java (cont)

```
        while (valeur != 0) //sentinelle 0 termine la boucle
        {
           compteur++;
           somme += valeur;
           System.out.println("La somme est " + somme);

           System.out.println("Entrez un entier (0 pour sortir)");
           valeur = Keyboard.readInt();
        }

        System.out.println("Nombre de valeurs entrées: " + compteur);
        moyenne = (double) somme/compteur;
        System.out.println("Moyenne des valeurs entrées: " + moyenne);
    }
}
```


Exécution de Moyenne.java

Entrez un entier (0 pour sortir) 67
La somme est 67
Entrez un entier (0 pour sortir) 23
La somme est 90
Entrez un entier (0 pour sortir) 4
La somme est 94
Entrez un entier (0 pour sortir) 0
Nombre de valeurs entrées: 3
Moyenne des valeurs entrées: 31.333333333333

Éviter les boucles sans fin

- Le corps de la boucle *while* doit, en fin de compte, s'assurer que la condition devient fausse
- Si non, c'est une boucle sans fin qui éxécutera jusqu'à ce que l'utilisateur arrête le programme
- Ceci est une erreur de logique typique
- Il est important de vérifier que vos boucles se terminent

Éviter les boucles sans fin: Toujours.java

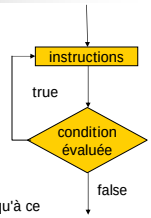
```
public class Toujours
{
   public static void main (String[] args)
   {
        int compteur = 1;

        while (compteur <= 25)
        {
               System.out.println(compteur);
               compteur--;
        }

        System.out.println("Fin");
    }
}
```


Les boucles imbriquées

- Semblable aux structures imbriquées *if*, les boucles peuvent être imbriquées aussi
- Le corps d'une boucle peut contenir une autre boucle
- Pour chaque itération de la boucle externe, la boucle interne exécute une itération complète
- Voir PalindromeTester.java (page 167)

Répétition: La structure *do*

```
do
{
   instructions ;
}
while ( condition )
```



- Les termes *do* et *while* sont réservés en Java
- Les *instructions* sont exécutées seulement une fois initialement, et ensuite la *condition* est évaluée
- Les *instructions* sont exécutées de façon itérative jusqu'à ce que la *condition* est fausse
- Note: si la condition d'une boucle while est fausse du départ, les instructions ne sont jamais exécutées
- Le corps de la boucle while sera exécuté zéro ou plus de fois

La structure *do*

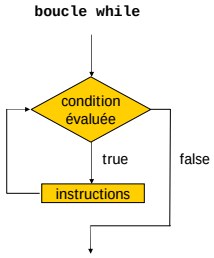
- La boucle *do* est semblable à la boucle *while*, apart que la condition est évaluée **après** que le corps de la boucle est exécuté
- Le corps de la boucle *do* sera exécuté au moins une fois

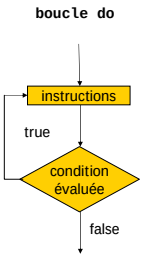
Quelle valeur est imprimé si compteur = 0 et LIMIT = 5?
Quelle valeur est imprimé si compteur = 5 et LIMIT = 5?
Quelle valeur est imprimé si compteur = 6 et LIMIT = 5?

```
do
{
   compteur = compteur + 1;
   System.out.println(compteur);
}
while (compteur < LIMIT);
```

```
while (compteur < LIMIT)
{
   compteur = compteur + 1;
   System.out.println(compteur);
}
```


Comparaison de while et do





Répétition: La structure *for*

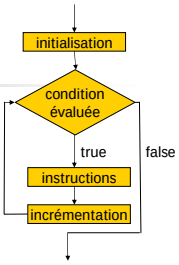
```
for ( initialisation ; condition ; incrémentation )
{
   instruction(s) ;
}
```

- Le terme *for* est réservé en Java
- L'initialisation est exécutée une fois avant que la boucle commence
- Les *instructions* sont exécutées jusqu'à ce la *condition* est fausse
- L'incrémentation est exécutée à la fin de chaque itération de la boucle
- Le cycle condition-instructions-incrémentation est exécuté à maintes reprises

Répétition: La structure *for*

- Une boucle *for* est équivalente en fonction à la structure *while* suivante:

```
initialisation;
while ( condition )
{
   instructions ;
   incrémentation;
}
```



- Comme la boucle *while*, la condition d'une boucle *for* est vérifiée avant d'exécuter le corps de la boucle
- Le corps de la boucle *for* sera exécuté zéro ou plus de fois

La structure *for* : Etoiles.java

```
*
**
***
****
*****
******
*******
********
*********
**********
```

```
public class Etoiles
{
   // Imprime des lignes d'étoiles, de 1 à 10

   public static void main (String[] args)
   {
        final int MAXLIGNES = 10;

        for (int ligne = 1; ligne <= MAXLIGNES; ligne++)
        {
               for (int etoile = 1; etoile <= ligne; etoile++)
                      System.out.println('*');
               System.out.println();
        }
   }
}
```


La structure *for*

- Chaque expression dans l'en-tête de la boucle *for* est optionnelle
  - Si l'*initialisation* est omise, aucune initialisation est effectuée
  - Si la *condition* est omise, elle est vraie par défaut, donc ceci crée une boucle sans fin
  - Si l'incrémentation est omise, aucune incrémentation est effectuée
- Les deux points-virgules sont toujours nécessaires dans l'en-tête de la boucle *for*

Choisir une structure de boucle: Quelques conseils

- Lorsqu'il est difficile de déterminer combien de fois vous voulez exécuter la boucle, utilisez la structure *while* ou *do*
  - Si zéro ou plus de fois, utilisez la boucle *while*
  - Si au moins une fois, utilisez la boucle *do*
- S'il est possible de déterminer le nombre d'itérations nécessaires, utilisez une structure *for*

49

Développement de logiciel

- La création de logiciel inclut quatre activités élémentaires:
  - collection des besoins
  - conception
  - implémentation
  - testing
- Le processus de développement est beaucoup plus complexe que ceci, mais ce sont les quatres activités de base

50

Développement de logiciel

- Supposons qu'on vous donne cette liste initiale de besoins:
  - accepte une série de notes d'examen
  - calcule la moyenne des notes d'examen
  - détermine le max et le min des notes d'examen
  - affiche la moyenne, le max et le min
- Discutez comment vous allez suivre les étapes de développment de logiciel pour créer une solution

51

Développement de logiciel: Analyse des besoins

- Clarifiez les besions
  - Quelle sera la quantité de données?
  - Comment accepter les données?
  - Y a-t-il un format exigé pour les données en sortie?
- À l'aide de discussion avec le client, on détermine que:
  - le programme doit traiter un nombre arbitraire de notes d'examen
  - le programme devrait accepter des données en mode interactif
  - la moyenne devrait être affiché avec deux places décimales
- Le processus d'analyse des besoins peut prendre beaucoup de temps

52

Développement de logiciel: Conception

- Déterminez une solution générale
  - Stratégie pour l'entrée des données? (valeur sentinelle?)
  - Calculs nécessaires?
- Un algorithme initial peut être exprimé en pseudo-code
- Il est possible que plusieurs versions de la solution seront créées avant que la conception soit raffinée
- Il faut considérer autres alternatives possibles pour la solution

53

Développement de logiciel: Implémentation

- Traduction de la conception en code source
  - Suivez les directrives de codage et de style
  - Intègrez l'implémentation à l'aide de compilation et de testing
- Ce processus réflète un modèle de développement plus complexe qui sera nécessaire pour un logiciel plus complexe
- Le résultat est une implémentation finale
- Voir la solution ExamScores.java (page 186)

54

Développement de logiciel: Testing

- Essayez de trouver des erreurs dans la solution programmée
  - Comparez le code avec la conception et résolvez les divergences identifiées
  - Créez des tests qui peuvent mettre à l'épreuve les limites de la solution
  - Soigneusement, re vérifiez la solution après avoir trouvé et réparé une erreur

55

Sommaire: Chapitre 3

- Comprendre le concepte de "flot de contrôle" dans une méthode
- Structure de sélection: *if*, *if-else*, et *switch*
- Comprendre comment utiliser les opérateurs
  - Opérateurs booléens: AND, OR, NOT
  - Autres opérateurs de Java: incrémentation ++, décrémentation --, assignation += et conditionnel ?
- Structure de répétition: *while*, *do*, et *for*
- Comprendre les étapes importantes du développement de logiciel

56