DEV2 - Lucky Summary

Sm!le42

14 juin 2021

Table des matières

1 (Array)List

2 Assignation

2.1 C'est quoi ?

L'*assignation* est avant tout une **expression**.
- Elle a un **type** (celui de la variable)
- Elle a une **valeur** (celle du *left-hand side*)

On en fait une **instruction** à l'aide du `";"`.

Grammaire de l'assignation :

Assignment :
LeftHandSide AssignmentOperator Expression
AssignmentOperator : one of `= *= /= %= += -=`

Exemple d'assignations :

```
element = 1       //Expression
elements[i] = j   //Idem

elements[i] = j; //Instruction
element += 2;    //Idem
foo(i=1, j=0);   //...
```

3 break/continue

3.1 C'est quoi ?

L'instruction `break` permet d'**arrêter brutalement** une instruction et de **sortir** de la boucle (ou du label).

L'instruction `continue` permet de **passer** directement à l'itération **suivante** (ou le label).

4 Collections

5 Constructeur

5.1 C'est quoi ?

Le *constructeur* d'une classe permet de créer des instances de cette classe :
- Lui **réserver de l'espace** en mémoire (sur le **tas**)
- **Initialiser son état** (ses attributs)

Exemple de constructeur de la classe Video :

```
public Video(String unAuteur, String unTitre) {
   this.auteur = unAuteur;
   this.titre = unTitre;
   this.publiee = false;
   this.nbLikes = 0;
   this.nbDislikes = 0;
}
```

5.2 new

Pour instancier un objet on va utiliser l'**opérateur** `new` et fournir d'éventuels **paramètres** au constructeur.

Exemple d'instanciation d'un objet **threadHorreur** de type **Video** :

```
Video threadHorreur = new Video("SQUEEZIE", "Êtes-vous vraiment seul chez vous ?");
```

5.3 Validité des paramètres

Le *constructeur* peut **vérifier** la *validité des paramètres*, il suffit d'ajouter des **test** au début de celui-ci.

Exemple de constructeur Video avec vérification de paramètres :

```java
public Video(String unAuteur, String unTitre) {
   if(unAuteur==null || unAuteur.length()==0) {
      throw new IllegalArgumentException("Auteur invalide");
   }
   if(unTitre==null || unTitre.length()==0) {
      throw new IllegalArgumentException("Titre invalide");
   }
   this.auteur = unAuteur;
   this.titre = unTitre;
   publiee = false;
   nbLikes = 0;
   nbDislikes = 0;
}
```

5.4 Valeurs par défaut

Si un constructeur n'**initialise pas** certain attributs, ils auront alors une *valeur par défaut* :

- `0` pour les nombres
- `null` pour les références
- `false` pour les booléens

5.5 Constructeur par défaut

Si nous n'écrivons **pas de constructeur**, il existe un *constructeur par défaut* **sans paramètre et qui ne fait rien**.

(Celui-ci ne sera plus disponible si un autre constructeur est fourni)

5.6 Plusieurs constructeurs (surcharge)

Il est possible de fournir *plusieurs constructeurs* **différents** pour une **même classe**. (Voir surcharge)

6 Encapsulation

6.1 C'est quoi ?

Le principe d'*encapsulation* permet de garder la cohérence de l'objet assurée par la classe.

- Les **attributs** sont **privés**
- Les **méthodes** permettant de **modifier l'état** de l'objet sont **publiques**

7 enum

8 equals

9 Expression régulière

9.1 Assignation

(Voir assignation)

9.2 Incrémentation/Décrémentation

Permet d'*incrémenter* ou de *décrémenter* une **variable**. (Voir PostIncrementation)

9.3 Appel de méthode

L'*appel de méthode* est une **expression**. Elle possède :
- un **type** (type du return)
- une **valeur** (valeur du return)

Exemple d'appel de méthode :

```
public class Foo {
   public static void main(String[] args) {
      double x = Math.sqrt(4);      //Type=double, valeur=2.0
   }
}
```

9.4 Instanciation de classe

Créer une instance d'une classe est une **expression**. Elle possède :
- Un **type** (celui de l'objet créé)
- Une **valeur** (la *référence* vers l'objet)

Exemple d'instanciation de classe :

```
public class Foo {
   public static void main(String[] args) {
      Video foo = new Video("auteur", "titre"); //Type=Video, Valeur=référence
   }
}
```

10 extends

11 Fichier texte/binaire

12 Filtrer (fonctionnel)

13 for

13.1 C'est quoi ?

L'instruction `for` permet d'effectuer du code en **boucle** un certain nombre de fois.

Elle est composée de la manière suivante :

`for (Initialisation ; Expression ; Update) {Instructions}`

13.2 Initialisation

Déclaration et **initialisation** de la variable utilisée pour **compter** le nombre de répétitions.

13.3 Expression

Test de l'expression qui retournera un **booléen** :
- `true` : Instructions - Update - ÉvaluationExpression (etc)
- `false` : On sort de la boucle

13.4 Update

Mise à jour la **valeur** de la **variable** initialisée lors de l'entrée dans la boucle.

14 foreach

14.1 C'est quoi ?

Permet de **parcourir** un Iterable.

Ainsi on peut **parcourir** une collection d'objets **sans devoir connaître le nombre** d'objets à parcourir.

Cette méthode est **plus rapide**, mais :
- **Pas** d'accès à l'indice
- **Impossible** de modifier un élément

Exemple d'instruction `foreach` :

```java
public void showAuteurs(ArrayList<Video> videos) {
   for(Video video : videos) {                //Pour chaque Video dans videos
      System.out.println(video.getAuteur());  //Affiche l'auteur de la vidéo
   }
}

//Équivalent avec un for classic:
public void showAuteurs(ArrayList<Video> videos) {
   for(int i = 0 ; i < videos.size() ; i++) {
      System.out.println(videos.get(i).getAuteur());
      //On a l'indice i donc on peut éventuellement modifier les données
   }
}
```

15 Grammaire

15.1 C'est quoi ?

La *grammaire* d'un langage est la **description** des **règles** de ce **langage**.
- Un **mot** (token) doit être légal (grammaire lexicale)
- Une **séquence de mots** doit être légale (grammaire syntaxique)
- Le **tout** doit avoir un sens (sémantique)

La *grammaire* du **Java** est décrite dans `The Java Language Specification`.

Chaîne de compilation : (Du haut vers le bas)

Programme source
`Analyse lexicale`
`Analyse syntaxique`
`Analyse sémantique`
Génération de code intermédiaire
Optimisation du code
Génération du code

15.2 Fonctionnement d'une grammaire

- Symbole de départ
- Règles de productions (*productions*)
- Symboles terminaux (*token*)

Un **code** est **correct** s'il peut être **produit** par la grammaire.

15.3 Grammaire lexicale

Des caractères aux mots.
- Les **symboles terminaux** sont les **caractères**
- Les **règles de production** forment les **mots** (*tokens*), **éléments d'entrée** (*inputElements*)

(Les commentaires et espaces ne passent pas la phase suivante)

15.4 Grammaire syntaxique

Des mots au programme.
- Les **symboles terminaux** sont les **tokens**
- Les **règles de production** permettent de définir ce qu'est un **programme syntaxiquement correct**

Parmi les **éléments importants d'un programme**, on retrouve :
- Les **expressions** (calculs, possèdent une valeur et un type)
- Les **instructions**

— Les **expressions-instructions**

Certaines **expressions** peuvent **devenir** une **instruction** dès l'ajout du `";"`.

15.5 Exemples de grammaire

Grammaire d'un nombre décimal naturel :

Nombre :
    Chiffre
    Chiffre Nombre
Chiffre : one of `0 1 2 3 4 5 6 7 8 9`

Grammaire d'un palindrome binaire :

Palindrome :
    `0`
    `1`
    `00`
    `11`
    `0` *Palindrome* `1`
    `1` *Palindrome* `0`

Grammaire de l'instruction `if` :

IfThenStatement :
    `if` (*Expression*) *Statement*
IfThenElseStatement :
    `if` (*Expression*) *StatementNoShortIf* `else` *Statement*
IfThenElseStatementNoShortIf :
    `if` (*Expression*) *StatementNoShortIf* `else` *StatementNoShortIf*

Grammaire de l'instruction `switch case` :

SwitchStatement :
    `switch` (*Expression*) *SwitchBlock*
SwitchBlock :
    { {*SwitchBlockStatementGroup*} {*SwitchLabel*}}
SwitchBlockStatementGroup :
    SwitchLabels BlockStatement
SwitchLabel :
    `case` *ConstantExpression* :
    `case` *EnumConstantName* :
    `default :`

Grammaire de l'instruction `while`

`while` (*Expression*) *Statement*

Grammaire de l'instruction `do-while`

`do` *Statement* `while` (*Expression*)

Grammaire de l'instruction `for` :

BasicForStatement :
    `for` *(ForInit ; Expression ; ForUpdate) Statement*
ForInit :
    StatementExpressionList
    LocalVariableDeclaration
ForUpdate :
    StatementExpressionList

Grammaire d'un `foreach` :

EnhancedForStatement :
    `for` (*Type Identifier* : *Expression*) *Statement*

16 if

16.1 C'est quoi ?

L'instruction `if` permet d'exécuter un certain code en fonction d'une **condition**.

Exemple de condition `if` } :

```java
public class Sign {
   public static void main(String[] args) {
      int foo = 0;
      System.out.println("Résultat du code:");

      //If-Then-Else
      if (foo < 0) {
         System.out.println("Foo est négatif");            //Ne sera pas exécuté
      } else {
         System.out.println("Foo est positif");
      }

      //If-Then-ElseIf-Else
      if (foo >= 0) {
         System.out.println("Foo est bien positif! *dab*");
      }
      else if (foo < 0) {
         System.out.println("Toujours pas négatif?");      //Ne sera pas exécuté
      } else {
         System.out.println("Heu.. Y a un problème ici!"); //Ne sera pas exécuté
         throw new UnexpectedException("wtf?");            //Ne sera pas exécuté
      }
   }
}
```

```
Résultat du code:
Foo est positif
Foo est bien positif! *dab*
```

16.2 If complexes

Lorsqu'un ensemble d'instructions `if` devient trop complexe et difficile à lire, on préférera utiliser l'instruction `switch-case`.

17 implements

18 import

19 Itérer (fonctionnel)

20 Object

20.1 Orienté objet

Un langage *orienté objet* permet de créer **ses propres types**, liés au problème à résoudre.

Avantages de l'*orienté objet* :
- lisibilité
- compactification
- robustesse

20.2 Un objet c'est quoi ?

Un *objet* est une **instance d'une classe** :
- construit à partir de la définition donnée par la classe
- appartenant au type défini par la classe

(Ex : Un objet `threadHorreur` pourrait être une instance de la classe `Video`)

20.2.1 Caractéristiques

1. **État** (**Données** de l'objet, stockées dans des **attributs**)

2. **Comportement** (Ce que l'on peut faire avec l'objet, en utilisant des **méthodes**)

Exemple : L'objet `threadHorreur` de la classe `Video` pourrait avoir les attributs et méthodes suivants :

threadHorreur : Video
–auteur="SQUEEZIE" –titre="Êtes-vous vraiment seul chez vous ?" –publiee=true –nbLikes=581356 –nbDislikes=4213
+liker() +disliker() +commenter()

20.3 Instancier un objet

Instancier un objet c'est le **construire en mémoire** à l'aide d'un **constructeur** :

— Lui **réserver de l'espace** en mémoire (sur le **tas**)

— **Initialiser son état** (ses attributs)

Exemple de constructeur de la classe Video :

```
public Video(String unAuteur, String unTitre) {
   this.auteur = unAuteur;
   this.titre = unTitre;
   this.publiee = false;
   this.nbLikes = 0;
   this.nbDislikes = 0;
}
```

20.3.1 new

Pour instancier un objet on va utiliser l'**opérateur** `new` et fournir d'éventuels **paramètres** au constructeur.

Exemple d'instanciation d'un objet **threadHorreur** de type **Video** :

```
Video threadHorreur = new Video("SQUEEZIE", "Êtes-vous vraiment seul chez vous ?");
```

20.4 Type référence

Une **classe** est un type *référence*. (Comme les tableaux)

```
Video threadHorreur;
//Référence créée sur la pile

threadHorreur = new Video("SQUEEZIE", "Êtes-vous vraiment seul chez vous ?";
//Objet créé sur le tas
```

21 Objects

22 Polymorphisme

23 Post-incrémentation

23.1 C'est quoi ?

La *post-incrémentation* lors de l'**évaluation d'une expression**, c'est lorsque cette variable est **incrémentée après** avoir donné sa valeur à l'expression.

Exemple de post-incrémentation avec ++ :

```java
public class PostIncr {
   public static void main(String[] args) {
      int i = 0;
      System.out.println("Résultat du code:");
      System.out.println("  i = " + i);
      System.out.println("i++ = " + (i++)); //Affiche i puis l'incrémente
      System.out.println("  i = " + i);
   }
}
```

```
Résultat du code:
  i = 0
i++ = 0
  i = 1
```

23.2 Pré-décrémentation

La *pré-décrémentation* est l'inverse de la *post-incrémentation*. La variable va donc être **décrémentée avant** de donner sa valeur à l'expression.

Exemple de pré-décrémentation avec – :

```java
public class PreDecr {
   public static void main(String[] args) {
      int i = 0;
      System.out.println("Résultat du code:");
      System.out.println("  i = " + i);
      System.out.println("--i = " + (--i)); //Décrémente i puis l'affiche
      System.out.println("  i = " + i);
   }
}
```

```
Résultat du code:
  i = 0
--i = -1
  i = -1
```

24 static

24.1 C'est quoi ?

Le mot-clé `static` permet de préciser qu'un **membre** fait référence à la **classe** (et non à une instance) et donc, celui-ci est **partagé** par toutes les instances.

24.2 Trois types de classes

En Java il existe *trois types de classes* :
- classe **utilitaire** (Ex : Math)
- classe **"objets"** (Ex : String, Scanner...)
- classe **mixte**

24.3 Membre static

Un *membre static* :
- fait référence à la **classe** (et non à une *instance*)
- est partagé par toutes les instances (éventuelles)

24.4 Attribut static

Un *attribut static* :
- existe en **un seul** exemplaire
- est **initialisé** lors du **chargement** de la classe (une seule fois)
- est souvent utilisé pour les **constantes**

Exemple d'attributs static :

```java
public class Math {
   public static final double PI = 3.141592;
   public static final double E = 2.718281;
}
```

24.5 Méthode static

Une *méthode static* :
- ne **peut pas accéder** aux membres des instances
- est souvent utilisée pour les méthodes **non objets**

Exemple de méthode static :

```java
public class Outils {
   public static int abs(int nb) {
      return nb < 0 ? -nb : nb; //Retourne la valeur absolue
   }
}
```

24.6 import static

Un *import static* créé un **raccourci** pour l'accès aux **membres statiques**.

Exemple d'import static :

```java
import static java.lang.Math.log;
import static java.lang.Math.E;

public class Test {
   public static void main(String[] args) {
      System.out.println(log(E));
   }
}
```

25 Surcharge/redéfinition

25.1 C'est quoi ?

Il est possible d'écrire **plusieurs fois la même méthode** en changeant le **nombre** ou le **type** de ses **paramètres**.

Par exemple on pourrait écrire une méthode `miser()` de **quatre manières différentes** :
- `miser()` qui mise 10€ au BlackJack par défaut
- `miser(int mise)` qui mise la mise voulue au BlackJack par défaut
- `miser(String jeu)` qui mise 10€ par défaut au jeu voulu
- `miser(int mise, String jeu)` qui mise la mise voulue au jeu voulu

Exemple des méthodes `miser()` :

```java
public void miser() {
   miser(10, "BlackJack");
}

public void miser(int mise) {
   miser(mise, "BlackJack");
}

public void miser(String jeu) {
   miser(10, jeu);
}

public void miser(int mise, String jeu) {
```

```
    //...
}
```

26 switch

26.1 C'est quoi ?

Un `switch case` est l'équivalent d'un ensemble de `if then - else if - else`.

Exemple d'instruction `switch case` :

```
public class Chaussettes {
    public static void main(String[] args) {
        int nbChaussettes = 2;

        System.out.println("Résultat du code:");
        switch(nbChaussettes) {
            case 3: //if (nbChaussettes == 3)
                System.out.println("Une de rechange au cas ou ;)");
                break;
            case 2: //else if (nbChaussettes == 2)
                System.out.println("Parfait, tu possèdes une paire.");
                break;
            case 1: //else if (nbChaussettes == 1)
                System.out.println("Tu as une seule chaussette? Pas très pratique..");
                break;
            case 0: //else if (nbChaussettes == 0)
                System.out.println("Zut.. Tu n'as pas de chaussettes");
                break;
            default: //else
                System.out.println("Tu as plus que 3 chaussettes apparemment...");
        }
    }
}
```

```
Résultat du code:
Parfait, tu possèdes une paire.
```

On utilise l'instruction `break` afin de sortir du `switch` sans exécuter ce qui suit.

S'il n'y avait aucun `break` dans le code précédant, le programme aurait exécuté tout ceci :

- `case 2`
- `case 1`
- `case 0`
- `default`

27 Tableau 1D

Un *tableau* est un **type de données** (de type référence).

On peut **créer** un tableau en fournissant :

- Les valeurs
- La taille

Exemple de quatre tableaux à une dimension :

```
public static void main(String[] args) {
    Video[] videos1;
    Video[] videos2 = new Video[3]; //Tout est initialisé à null
    videos1 = new Video[3];         //Tout est initialisé à null

    videos1[0] = new Video("auteur1", "titre1");
    videos1[1] = new Video("auteur2", "titre2");
    videos2[0] = new Video("auteur3", "titre3");
```

```
    videos2[1] = new Video("auteur4", "titre4");

    Video[] videos3;
    Video[] videos4 = /*new Video[]*/ {new Video("auteur5", "titre5"),
                                       new Video("auteur6", "titre6"),
                                       null};
    videos3 = new Video[] {new Video("auteur7", "titre7"),
                           new Video("auteur8", "titre8"),
                           null};
}
```

28 Tableau 2D

28.1 C'est quoi ?

Un *tableau 2D* n'est rien d'autre qu'un tableau de tableau.

Exemple de trois tableaux à deux dimensions :

```
public static void main(String[] args) {
    Video[][] videos;
    videos = new Video[][] {{new Video("aut1", "titr1"), new Video("aut2", "titr2")},
                            {new Video("aut3", "titr3"), new Video("aut4", "titr4")}};
    int[][] pascal = {{1},
                      {1, 1},
                      {1, 2, 1},
                      {1, 3, 3, 1},
                      {1, 4, 6, 4, 1}};

    int[][] sudoku = new int[9][9]; //Tout est initialisé à 0
}
```

29 this/super

29.1 This

This est une **référence à soi-même**.

Elle apparaît dans différents contextes :
- Constructeur `this()`
- Attributs `this.auteur`
- Méthodes `this.liker()`

Exemple d'utilisation du mot-clé `this` :

```
public Video(String unAuteur, String unTitre, boolean publiee) {
    this.auteur = unAuteur;       //"this" facultatif car auteur != unAuteur
    this.titre = unTitre;           //Idem
    this.publiee = publiee;       //"this" important car publiee == publiee
    this.nbLikes = 0;             //"this" facultatif car nbLikes est unique ici
    this.nbDislikes = 0;          //Idem
}

public Video(String unAuteur, String unTitre) {
    this(unAuteur, unTitre, false); //Doit être la 1ère instruction!
}

public void liker() {
    this.addLike();
    System.out.println("Vous avez liké la vidéo");
}

```

```java
private void addLike() {
   this.nbLikes++;
}
```

30 throw(s)

31 toString

31.1 C'est quoi ?

La méthode `String toString()` :
- fournit une représentation **textuelle basique** de l'état
- a un **nom standardisé**
- est **appelée automatiquement** par `println` ou lors de concaténation
- une **version par défaut** existe, mais n'est **pas intéressante**

Exemple d'utilisation de la méthode toString() :

```java
public String toString() {
   return "Auteur: " + this.auteur
        + "Titre: " + this.titre
        + "Est publiée: " + this.publiee
        + "Nombre de likes: " + this.nbLikes
        + "Nombre de dislikes: " + this.nbDislikes;
}
```

32 Trier

33 try-catch

34 var

35 Var args

36 Visibilité

36.1 C'est quoi ?

En Java, chaque **membre** possède un des **quatre types** de visibilité suivants :
- `public` (Accessible depuis **toutes les classes**)
- `private` (Accessible uniquement depuis la **classe**)
- `package` (Accessible depuis le **package**)
- `protected` (Accessibilité liée à l'**héritage**)

37 while/do while

37.1 C'est quoi ?

Les instructions `while` et `do while` permettent d'effectuer des **boucles** qui s'exécuteront **tant que** l'expression est **vraie**.

Contrairement aux boucles `for`, on peut utiliser les `while` et `do while` pour effectuer du code à répétition **sans connaître** le nombre de fois à l'avance.

37.2 Différence entre `while` et `do while`

- L'instruction `while` va d'abord **vérifier** si l'expression est **vraie**, puis éventuellement **exécuter** le code.
- L'instruction `do while` va d'abord **exécuter** le code, puis **vérifier** si l'expression est **vraie**.

Exemple de boucles `while` et `do while` :

```java
public class HelloWorldx7 {
   public static void main(String[] args) {
      int i = 0;

      System.out.println("Résultat du code:");
      while (i++ < 3) { //Incrémentation après évaluation de l'expression
         System.out.println("Hello world!");
      }

      System.out.println();

      do {
         System.out.println("Hello world!");
      } while (--i > 0); //Décrémentation avant évaluation de l'expression
   }
}
```

```
Résultat du code:
Hello world!
Hello world!
Hello world!

Hello world!
Hello world!
Hello world!
Hello world!
```

(Pour comprendre le fonctionnement de `i++` et `--i`, voir post-incrémentation)

38 Wrapper/boxing