livres : - Java pour les nuls

- scholarvox.com

lire : algo et types de base (cyberlearn)

Raccourci Intellij : shift + shift : recherche tout

ctrl + alt + t : surround with template

alt + p : all occurence

ctrl + shift + enter : finish if, for, ...

completion → (){}

ctrl + shift + n : " fichier

ctrl + n : " class

ctrl + w : expand selection

shift + alt + click : multiple carret

molette click déplacement : " "

un/commenter : shift + §

ctrl + d : delete line(s)

ctrl + y : duplicate line(s)

alt + arrow : move line(s)

ctrl + alt + arrow : move method

- Package est un namespace = dossier
  ↓ contient
  Class sont dans des fichiers

- 2 types de class : - utilitaires : comme String
  - modèle : pour instancier des objets créés par dev.

- Aucune logique dans la méthode main, que des appels
- Aucun code n'est en dehors de class / méthod / fx
- Module c'est un ensemble de fichier associés à un nom, comme un package mais de taille réduite. Version améliorée.
- Niveau de contrôle :
  1. public : accessible à tous
  2. protected : accessible dans la classe et ses enfants
  3. private : accessible uniquement dans la class, class enfants n'héritent pas.

le Polymorphisme peut changer cela mais que les enfants peuvent le faire.

- for (initialisation, terminaison, incrémentation) {

  ...

  }

- tableau / collection

  ↳ int[] myArray = new int[]{7, 2, 4};

  for (int i=0 ; myArray.length ; i++){ ... }

"foreach" = for (int i : myArray){ ... }

- Voir les 5 slides de casting...

- while (x < 10){

  x++;

  ...

  }

- do {

  x++;

  } while (x < 10)

} le bloc est exécuté au moins une fois

⚠ - On peut utiliser le break pour sortir d'une boucle.

```
class Book {

    String title;
        ⋮
    Book (String title, ...) {
        this.title = title;
        ⋮
    }

    Book (...) {
        this (....)
    }

}
```

constructeur principale

constructeur secondaire

- héritage avec extends
- tous les attributs sont hérités sauf private leurs manipulation dépend de leur accessibilité.
- utiliser le terme super(...) pour accéder au constructeur

- Polymorphisme : possibilité de redéfinir certaine méthode de la classe mère

```
Class ... extends {

    @Override
    void start(){
        super.start();
        uneAutreMethode();
    }
```

⚠ annotations servent à spécifier le comportement d'une classe, méthode, attr., var, ...

- énumération : voir sur wikibooks ...

```
public enum Polygone {
    TRIANGLE(3), PENTAGONE(5), ...
    private final int nbCote;
    private Polygone
        :
}
```

permet au compilateur de comparer la signature de la classe parente

- Les arrays pour gérer un nombre fixe d'éléments

```
int [] monEntier = { 1, 2, 3 ... }
```

index 0, ...

- on peut définir la longueur d'un tableau

```
int [] limitedArray = new int [7];
```

les valeurs par défauts sont que de 0 pour des ints.

- on peut retourner une valeur en array :

```
return new int [] {6, 2, 3, ...};
```

- Tableau multidimensionels : `[][]`

row size, column size

⚠ Arrays = taille fixe

- Les collections, pour gérer une taille var. d'éléments
  - plusieurs types :

1. La liste : ArrayList = ordrée

```
List <String> mylist = new ArrayList<String>();
```

c'est une interface — c'est le paramètre de type

permet de limiter le type d'objet dans la liste. Donc si on ne mentionne pas on peut mettre n'import quel type, le problème est que on devra caster/convertir afin de l'utiliser.

```
List <Integer> mylist = new ArrayList<Integer>();
```

en majuscule ? Oui les liste ne peuvent que stocker des objets. Donc pas de type primitif.

// Une interface est un contrat qui définit toutes les opérations qu'une classe doit fournir

* Le boxing est une conversion par le compilateur de type primitif à son type objet.

* Le méthodes pour l'interface List :

.add (13) -> ajoute

.add (2, 13) -> insère a l'index 2 le 13 et déplace la valeur existante à la suivante ...

.set (0, 4) -> définit la valeur à un index donc l'écrase.

.remove (1) -> supprime l'index.

.size () -> .length

2. Une collection non-ordonnée (Set) (ie: ingrédient de cuisine)

Un ensemble ou "set" est une collection d'éléments uniques non ordonnés.

`Set<String> ingredients = new HashSet<String>();`

Set<String> : type + param de type (interface)

HashSet<String>() : type d'ensemble + param de type

↳ add(), remove("salt"), size(), ~~set()~~

3. Dictionnaire : clé - valeur : Map interface

```
Map <String, Integer> myMap = new HashMap <Str., Int.>();
```

clé — valeur — type de dictionnaire

méthodes :

- `.put("Nom", "Joa");` permet d'ajouter
- `.get("Nom");` renvoie la valeur de la clé
- `.remove("x");` ...
- `.size();` == `.length()`

4. Résumé :

- Conteneurs de taille fixe : Arrays

collection :

- Listes ordonnées : List (ArrayList)
- Listes non-ordonnées : Set (HashSet)
- Dictionnaires clé-valeur : Map (HashMap)

Différencier type valeur et type référence (pointeur)

- type valeur :

  Les variables/constantes de type primitif sont de type valeur, cela veut dire que si l'on a :

```
int x = 1;
int y = x;
x = 2;
sout(y); == 1
```

  Pourquoi ? parce que les types valeurs créé un nouvel emplacement en mémoire et chaque variable primitive est indépendante.

- type référence ou pointeur donc même instance :

  Lorsqu'on créé une variable et qu'on lui affecte une instance de classe, l'objet est créé, mais la variable contient uniquement la référence à cet objet donc l'emplacement en mémoire. Et lorsqu'on copie cette variable à une autre, l'autre variable contiendra uniquement la référence.

Exemple :

```
int cote = 5;
agrandirCote(cote);
```

```
Integer cote = 5;
agrandirCote(cote);
```

```
protected static int/Integer agrandirCote(int/Integer cote)
{
    cote += 2;
}
```

sout(cote); ? = 5 ou 7

Récursivité

$5! = 5 \times 4 \times 3 \times 2 \times 1$

$N! = N \times (N-1)!$

$5! = 5 \times (5-1)!$

$5! = 5 \times 4!$

$5! = 5 \times 4 \times (4-1)!$

$\vdots$

} = factorielle

En java :

```
psi factorial (int n) {
    if (n==1) return 1;
    else return n * factorial (n-1);
}
```

Voir un code propre de récursivité en pdf...

Gérer les exceptions/erreurs

Deux manières de gérer celles-ci :

1. En utilisant le conditions

```
if ( arrayInt.length() != 0) {

    ok poursuivre ...

}
```

2. En utilisant try - catch

```
try {
    :
} catch (NumberFormatExc... e){
    sout ("...");
    System.exit(-1);
}
catch (ArithmeticException e) {
    sout("...");
    System.exit(-1);
}
```

mais on peut utilise Exception pour attraper tout type d'erreur.

Manipuler les fichiers

~ lire un fichier

```
FileReader fileReader = new FileReader("/path/...");
```

↑ Permet de définir où lire

```
BufferedReader reader = new BufferedReader(fileReader);
```

↑ permet de lire ligne par ligne un fichier

```
try {
    line = reader.readLine();
    while (line != null){
        sout(line);
        line = reader.readLine();
    }
} catch (IOException){
    e.printStackTrace();
}
reader.close();
```

- écrire dans un fichier

```
FileWriter fileWriter = new FileWriter("/path/...", false);
BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
```

le paramètre false précise que si des données sont déjà existante dans le fichier alors on écrase.

Même syntaxe que le lecteur...

* utiliser la méthode .newLine(); pour un retour à la ligne.
* utiliser la méthode .write("..."); pour écrire.

```
:
writer.close();
```

Fx Lamda = Fx anonyme (closure)

- interface fonctionnelle : celle-ci n'auront qu'une seule méthode abstraite.

```
()-> action
(arg1, arg2 ...) -> action
(arg1, arg2 ...) -> { ... };
```

```
public interface Etudiant {
    void donnerNom(String nom);
}
psv main (String[] args) {
    Etudiant et =(nom)-> { sout("..." + nom)};
    et.donnerNom("Joe");
}
```

Concepts avancés à connaître :

* Classe générique
* Interface générique
* Interface fonctionnelle (lambda)
* Expressions lambda
* Streams
* Méthodes par défaut et extensions
* Annotations
* Réflexions
* Gestion des exceptions
* Threads
* Synchronisation
* JPA
* JAX-RS : services Web RestFul