MODULARISATION

DÉCOMPOSER UN PROGRAMME EN MÉTHODES

RAPPEL - STRUCTURE D'UN PROGRAMME JAVA

```java
public class NomDeClasse {
  public static void main(String[] args) {
    instruction;
    instruction;
    ...
    instruction;
  }
}
```

- Tout programme Java contient (au moins) une **classe**
  - qui contient une **méthode** appelée `main`
    - qui contient des **instructions** à exécuter

RAPPEL - ALGORITHME

- ⇒ description de la résolution étape par étape d'un problème
- Ex : *cuisiner des cookies*

```
- Préchauffer four 180°c
- Mélanger les ingrédients secs
- Mélanger beurre + sucre
- Intégrer beurre + sucre
- Battre les blancs en neige
- Intégrer blancs en neige
- Faire des boules aplaties
- Les placer dans le four
- Attendre 15 minutes
- Sortir les cookies
- Mélanger les ingrédients pour la garniture
- Appliquer la garniture sur les cookies
```

STRUCTURE

- Cet algorithme :
  - **manque de structure** : juste une longue liste d'étapes
  - **est redondant** : que se passe-t-il si on veut faire la cuisson en 2 fournées ?
    - ⇒ on doit recopier le code de cuisson
- La redondance induit des problèmes (**duplication**) :
  - la cuisson change ? ⇒ il faut modifier les deux parties
  - une troisième fournée ? ⇒ il faut encore recopier tout le traitement

ALGORITHME STRUCTURÉ

Pour régler ces problèmes de structure et de duplication, on va **modulariser** l'algorithme, c'est-à-dire le découper en sous-problèmes

Pour cela, il faut appliquer la procédure suivante :

- Identifier les sous-problèmes
- Nommer les sous-problèmes
- Extraire les sous-problèmes

IDENTIFICATION DES SOUS-PROBLÈMES

Il faut **décomposer** l'algorithme de manière cohérente en gardant ensemble les parties qui sont au même niveau d'abstraction et qui constituent un sous-problème complet

```
- Préchauffer four 180°c
- Mélanger les ingrédients secs
- Mélanger beurre + sucre
- Intégrer beurre + sucre
- Battre les blancs en neige
- Intégrer blancs en neige
- Faire des boules aplaties
- Les placer dans le four
- Attendre 15 minutes
- Sortir les cookies
- Mélanger les ingrédients pour la garniture
- Appliquer la garniture sur les cookies
```

```
- Préchauffer four 180°c
-----------------------------------------
- Mélanger les ingrédients secs
- Mélanger beurre + sucre
- Intégrer beurre + sucre
- Battre les blancs en neige
- Intégrer blancs en neige
- Faire des boules aplaties
-----------------------------------------
- Les placer dans le four
- Attendre 15 minutes
- Sortir les cookies
-----------------------------------------
- Mélanger les ingrédients pour la garniture
- Appliquer la garniture sur les cookies
```

NOMMAGE DES SOUS-PROBLÈMES

On **nomme** chaque sous-problème (le choix des noms est **très important** pour la lisibilité et la compréhension)

```
- Préchauffer four 180°c
-----------------------------------------
- Mélanger les ingrédients secs
- Mélanger beurre + sucre
- Intégrer beurre + sucre
- Battre les blancs en neige
- Intégrer blancs en neige
- Faire des boules aplaties
-----------------------------------------
- Les placer dans le four
- Attendre 15 minutes
- Sortir les cookies
-----------------------------------------
- Mélanger les ingrédients pour la garniture
- Appliquer la garniture sur les cookies
```

```
- Préchauffer four 180°c
- RÉALISER L'APPAREIL
  - Mélanger les ingrédients secs
  - Mélanger beurre + sucre
  - Intégrer beurre + sucre
  - Battre les blancs en neige
  - Intégrer blancs en neige
  - Faire des boules aplaties
- CUIRE LES COOKIES
  - Les placer dans le four
  - Attendre 15 minutes
  - Sortir les cookies
- DÉCORER LES COOKIES
  - Mélanger les ingrédients pour la garniture
  - Appliquer la garniture sur les cookies
```

EXTRACTION DES SOUS-PROBLÈMES

- On **extrait** chaque sous-problème en dehors de l'algorithme principal

```
- Préchauffer four 180°c
- RÉALISER L'APPAREIL
  - Mélanger les ingrédients secs
  - Mélanger beurre + sucre
  - Intégrer beurre + sucre
  - Battre les blancs en neige
  - Intégrer blancs en neige
  - Faire des boules aplaties
- CUIRE LES COOKIES
  - Les placer dans le four
  - Attendre 15 minutes
  - Sortir les cookies
- DÉCORER LES COOKIES
  - Mélanger les ingrédients pour la garniture
  - Appliquer la garniture sur les cookies
```

```
Sous-problème "RÉALISER L'APPAREIL"
- Mélanger les ingrédients secs
- ...

Sous-problème "CUIRE LES COOKIES"
- Les placer dans le four
- ...

Sous-problème "DÉCORER LES COOKIES"
- Mélanger les ingrédients pour la garniture
- ...

Problème principal "CUISINER DES COOKIES"
- Préchauffer four 180°c
- Appel de "RÉALISER L'APPAREIL"
- Appel de "CUIRE LES COOKIES"
- Appel de "DÉCORER LES COOKIES"
```

MODULARISATION

```
- Problème "CUISINER DES COOKIES"
  - Préchauffer four 180°c
  - Appel de "RÉALISER L'APPAREIL"
  - Appel de "CUIRE LES COOKIES"
  - Appel de "DÉCORER LES COOKIES"
```

L'algorithme général est maintenant bien plus lisible

- on a une vision de plus haut niveau de ce que fait l'algorithme
- si on a besoin des détails, on examine les algorithmes correspondants aux sous-problèmes

Qu'en est-il des problèmes de duplication ?

EXEMPLE DE DUPLICATION

```
 1 - Préchauffer four 180°c
 2 - Mélanger les ingrédients secs
 3 - Mélanger beurre + sucre
 4 - Intégrer beurre + sucre
 5 - Battre les blancs en neige
 6 - Intégrer blancs en neige
 7 - Faire des boules aplaties
 8 - Placer le 1er fournée dans le four
 9 - Attendre 15 minutes
10 - Sortir les cookies
11 - Placer le 2ème fournée dans le four
12 - Attendre 15 minutes
13 - Sortir les cookies
14 - Mélanger les ingrédients pour la garnitu
15 - Appliquer la garniture sur les cookies
```

```
 1 - Préchauffer four 180°c
 2 - Mélanger les ingrédients secs
 3 - Mélanger beurre + sucre
 4 - Intégrer beurre + sucre
 5 - Battre les blancs en neige
 6 - Intégrer blancs en neige
 7 - Faire des boules aplaties
 8 - Cuire les cookies (1ère fournée)
 9 - Cuire les cookies (2ème fournée)
10 - Mélanger les ingrédients pour la garnitu
11 - Appliquer la garniture sur les cookies
```

LA DÉCOMPOSITION ÉLIMINE LA DUPLICATION

La duplication est quasiment éliminée

- ici, une boucle de 2 itérations appelant **Cuire les cookies** éliminerait complètement la duplication
- mais les boucles ne permettent pas d'éliminer la duplication systématiquement (ex. : code dupliqué ne se trouvant à des endroits différents dans le code)

Avec la décomposition :

- la cuisson change ? ⇒ un seul endroit à modifier
- une troisième fournée ? ⇒ on ajoute un appel à **Cuire les cookies**

DÉCOMPOSITION PROCÉDURALE

Ce mécanisme s'appelle la **décomposition procédurale**

- ⇒ séparation d'une problème en plusieurs sous-problèmes
- ⇒ chaque sous-problème est plus simple que le problème englobant

Ces sous-problèmes peuvent eux-mêmes être décomposés en d'autres sous-problèmes au besoin

Ex : **Battre les blancs en neige** peut être décomposé :

```
Sous-problème "BATTRE LES BLANCS EN NEIGE"
- Séparer les jaunes des blancs
- Battre 30 secondes au fouet
- Ajouter du sel
- Battre jusqu'à consistance ferme
```

DÉVELOPPEMENT ITÉRATIF

Le **développement itératif** consiste à écrire un programme par étapes, en ajoutant une fonctionnalité à la fois

L'un des grands bénéfices que l'on en tire est la capacité à **tester chaque partie au fur et à mesure** pour s'assurer que tout fonctionne bien avant de continuer

Java permet d'appliquer la **décomposition procédurale** et le **développement itératif** grâce notamment aux **méthodes**

MÉTHODES

Une **méthode** désigne un bloc d'instructions auquel on donne un nom

- un peu comme une « commande » Java qu'on écrit nous-mêmes
- ⇒ sous-problème qu'on a identifié et qu'on va isoler
- permet la décomposition (**Faire l'appareil**, **Cuire les cookies**, **Décorer les cookies**…)
- supprime la redondance (chaque « commande » pourra être répétée à volonté sans duplication de code)

PROCESSUS DE DÉCOMPOSITION

Concevoir l'algorithme

- Repérer la structure de la résolution du problème
  - Quels sont les sous-problèmes importants ? Peuvent-ils eux-mêmes être décomposés en sous-problèmes ?
  - Dans tout cela, y a-t-il des choses qui sont répétées ?

Définir (écrire) les méthodes

- Chaque sous-problème identifié sera dans sa propre méthode bien nommée

Appeler (invoquer) les méthodes

- Dans la méthode `main` mais aussi dans n'importe quelle autre méthode si celle-ci est elle-même décomposée

DÉFINIR UNE MÉTHODE

```
public static void maMéthodeSuperBienNommée() {
  // instructions
}
```

Convention de nommage : `camelCase` (commence par une minuscule)

`void` est le **type de retour** : cette méthode ne renvoie rien à l'appelant (*void* = vide)

- on appelle *appelant* la méthode qui a appelé la méthode courante
- on verra qu'il est souvent utile de renvoyer une valeur à l'appelant

L'ensemble d'instructions correspondantes à la méthode se trouve dans le bloc d'accolades { ... }

Même définie, **une méthode n'est pas exécutée tant qu'elle n'est pas appelée**

APPELER (INVOQUER) LA MÉTHODE

```
// Appel de méthode (exécution)
maMéthodeSuperBienNommée();
```

Appel de méthode : instruction qui provoque l'*exécution* de la méthode appelée

- ce qui cause l'exécution de la *{ séquence d'instructions }* de cette méthode

Quand la méthode est terminée, le contrôle est rendu à l'appelant

C'est comme si on commandait à l'ordinateur : « *À chaque fois que je te dis* `maMéthodeSuperBienNommée` *, je veux que tu exécutes la séquence d'instructions qui se trouve dans cette méthode* »

Une fois définie, une méthode peut être appelée autant de fois qu'on veut

Appeler une méthode non définie entraîne une erreur de compilation

APPELER LA MÉTHODE - SYNTAXE

```
// Appel de méthode (exécution)
maMéthodeSuperBienNommée();
```

Parenthèses obligatoires

Point-virgule obligatoire (un appel de méthode est une instruction)

Comme toujours, attention à la casse

Faites confiance à l'*auto-complétion* de votre IDE pour faire respecter la casse et l’orthographe

DÉFINITION ET APPELS DE MÉTHODES - EXEMPLES

```java
public class WeWillRockYou {
  public static void main(String[] args) {
    System.out.println("We will, we will, rock you!");
    System.out.println();
    System.out.println("We will, we will, rock you!");
  }
}
```

```java
public class WeWillRockYou {
  public static void main(String[] args) {
    sing();
    System.out.println();
    sing();
  }

  public static void sing() {
    System.out.println("We will, we will, rock you!");
  }
}
```

```java
public class CuisinerCookies {

  public static void main(String[] args) {

    System.out.println("Mélanger les ingrédients secs");
    System.out.println("Mélanger beurre + sucre");
    System.out.println("Intégrer beurre + sucre");
    System.out.println("Battre les blancs en neige");
    System.out.println("Intégrer blancs en neige");
    System.out.println("Faire des boules aplaties");
    System.out.println("Placer la 1ère fournée dans le four");
    System.out.println("Attendre 15 minutes");
    System.out.println("Sortir les cookies");
    System.out.println("Placer la 2ème fournée dans le four");
    System.out.println("Attendre 15 minutes");
    System.out.println("Sortir les cookies");
    System.out.println("Mélanger les ingrédients pour la garniture");
    System.out.println("Appliquer la garniture sur les cookies");

  }
}
```

```java
public class CuisinerCookies {

  public static void main(String[] args) {

    // Réaliser l'appareil
    System.out.println("Mélanger les ingrédients secs");
    System.out.println("Mélanger beurre + sucre");
    System.out.println("Intégrer beurre + sucre");
    System.out.println("Battre les blancs en neige");
    System.out.println("Intégrer blancs en neige");
    System.out.println("Faire des boules aplaties");

    // Cuire fournée 1
    System.out.println("Placer la 1ère fournée dans le four");
    System.out.println("Attendre 15 minutes");
    System.out.println("Sortir les cookies");

    // Cuire fournée 2
    System.out.println("Placer la 2ème fournée dans le four");
    System.out.println("Attendre 15 minutes");
    System.out.println("Sortir les cookies");

    // Décorer les cookies
    System.out.println("Mélanger les ingrédients pour la garniture");
    System.out.println("Appliquer la garniture sur les cookies");

  }
}
```

```java
public class CuisinerCookies {
  public static void main(String[] args) {
    RéaliserAppareil();
    CuireCookies();
    CuireCookies();
    DécorerCookies();
  }
}
public static void RéaliserAppareil() {
  System.out.println("Mélanger les ingrédients secs");
  System.out.println("Mélanger beurre + sucre");
  System.out.println("Intégrer beurre + sucre");
  System.out.println("Battre les blancs en neige");
  System.out.println("Intégrer blancs en neige");
  System.out.println("Faire des boules aplaties");
}
public static void CuireCookies() {
  System.out.println("Placer les cookies dans le four");
  System.out.println("Attendre 15 minutes");
  System.out.println("Sortir les cookies");
}
public static void DécorerCookies() {
  System.out.println("Mélanger les ingrédients pour la garniture");
  System.out.println("Appliquer la garniture sur les cookies");
}
```

MÉTHODES - SYNTHÈSE (1)

Les méthodes rendent le code **plus facile à lire** : elles permettent de **décomposer le programme** et d'en **capturer la structure**

Les **noms de méthodes efficaces** renforcent cette qualité

La méthode `main` devrait être un **bon résumé** de ce qui se passe dans le programme

Plus généralement, *toute méthode* devrait se lire relativement rapidement

- les détails sont relégués aux méthodes de plus bas niveau (les sous-problèmes)

MÉTHODES - SYNTHÈSE (2)

Modulariser un code redondant permet :

- d'avoir **un seul point de modification** pour ce code (et facile à retrouver)
- de pouvoir **répéter** à volonté l'action décrite (pas de copier/coller !)

Parfois, décomposer en méthodes rend le code plus long, surtout s'il n'y avait pas de redondance à éliminer

- Mais **un code plus long ne signifie pas nécessairement un code de plus mauvaise qualité**

QUAND NE PAS UTILISER DE MÉTHODES

Ne créez pas de méthodes pour :

- exécuter des instructions ultra-simples (genre un `println` vide)
- regrouper des traitements qui ne sont pas vraiment en relation (mauvais décomposition)

FLUX D'EXÉCUTION

Le flux d'exécution est comme un « curseur » qui se déplace d'instruction en instruction

Ce curseur pointe toujours sur la prochaine instruction qui va être exécutée

Au lancement du programme, l'environnement d'exécution place le curseur sur la première instruction de la méthode `main`

Quand une instruction est terminée, le curseur passe à la suivante

Le programme se termine lorsque le flux d'exécution atteint la fin de la méthode `main`

FLUX D'EXÉCUTION - APPELS DE MÉTHODES

Quand une méthode est appelée, le curseur :

- « saute » dans la méthode concernée, sur la première instruction
- suit le flux des instructions de la méthode
- et « re-saute » au point où la méthode a été invoquée (*retour à l'appelant*)

Le flux d'exécution peut ainsi suivre un tunnel d'appels imbriqués de méthodes, potentiellement très profond

- mais le curseur sait toujours d'où il vient

COMPRENDRE LE FLUX D'EXÉCUTION

```
public class Omelette {
  public static void main(String[] args) {
    BattreLesOeufs();
    CuireOmelette();
  }

  public static void BattreLesOeufs() {
    PrendreLesOeufs();
    CasserLesOeufsDansBol();
    Assaisonner();
    System.out.println("Battre les oeufs avec une fourchette");
  }

  public static void PrendreLesOeufs() {
    System.out.println("Ouvrir le placard en haut à droite");
    System.out.println("Prendre la boîte à oeufs");
    System.out.println("Fermer le placard en haut à droite");
  }

  // Autres méthodes non incluses...
}
```

QUELLE EST LA SORTIE ?

```java
public class QuelleEstLaSortie {
  public static void foo() {
    System.out.println("chouette");
    baz();
  }

  public static void bar() {
    baz();
    System.out.print("machin ");
    foo();
  }

  public static void baz() {
    System.out.println("truc");
  }

  public static void main(String[] args) {
    foo();
    bar();
    System.out.println("bidule");
  }
}
```

SOLUTION

- Voici l'ordre dans lequel les méthodes sont appelées :
  - *main*, *foo*, *baz*, *bar*, *baz*, *foo*, *baz*
- Et voici la sortie :

```
chouette
truc
truc
machin chouette
truc
bidule
```