

# Compléments en Programmation Orientée Objet

## Aldric Degorre

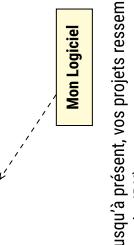
Version 2019-10 - 00 du 28 novembre 2019

**Chargeés de TP en 2019 :**  
Isabelle Fagnot<sup>1</sup> (jeudi), Yan Jurski<sup>2</sup> (mardi), Yixin Shen<sup>3</sup> (jeudi) et Aldric Degorre<sup>4</sup> (lundi).  
*En remerciant mes collaborateurs des années passées, qui ont aidé à élaborer ce cours et à le faire évoluer.*

---

1. fagnot@irif.fr  
2. jurski@irif.fr  
3. yixin.shen@irif.fr  
4. adegorre@irif.fr

**On a déjà « fait Java », pourquoi ce cours ?**  
La vraie vie ?



Jusqu'à présent, vos projets ressemblaient à ça (un logiciel exécutable qui ne dépend que du JDK).

**On a déjà « fait Java », pourquoi ce cours ?**  
La vraie vie ?

De plus, tous ces logiciels évoluent au cours du temps (versions successives).

**Comment s'assurer alors :**

- que votre programme résiste aux évolutions de ses dépendances ?
- qu'il fournit bien à ses clients le service annoncé, dans toutes les conditions annoncées comme compatibles<sup>7</sup>
- que les évolutions de votre programme ne « cassent » pas ses clients ?

Pensez-vous que le projet rendu l'année dernière garantit tout cela ?

---

7. Il faut essayer de penser à tout. Notamment, la bibliothèque fonctionnel·le comme prévu quand elle est utilisée par plusieurs threads ?

<b>Compléments en POG</b>	<b>Aidez l'élève à</b>	<b>Compléments en POG</b>	<b>Aidez l'élève à</b>
<b>Introduction</b>	<b>Généralités</b>	<b>Introduction</b>	<b>Généralités</b>
<b>Classes</b>	<b>Style</b>	<b>Classes</b>	<b>Style</b>
<b>Types et polymorphisme</b>	<b>Objets et classes</b>	<b>Types et polymorphisme</b>	<b>Objets et classes</b>
<b>Héritage</b>	<b>Interfaces graphiques</b>	<b>Héritage</b>	<b>Interfaces graphiques</b>
<b>Généricité</b>	<b>Gestion des erreurs et exceptions</b>	<b>Généricité</b>	<b>Gestion des erreurs et exceptions</b>
<b>Concurrency</b>		<b>Concurrency</b>	
<b>Interfacing</b>		<b>Interfacing</b>	
<b>Gestion des erreurs et exceptions</b>		<b>Gestion des erreurs et exceptions</b>	

## On a déjà « fait Java », pourquoi ce cours

Sauriez-vous encore ?

- Supprimer un bug, ce fameux bug que vous n'aviez pas eu le temps de corriger avant la deadline de P14 l'année dernière ?
- Remplacer une des dépendances de votre projet par une nouvelle bibliothèque, sans tout modifier et ajouter 42 nouveaux bugs ?
- Ajouter une extension que vous n'aviez pas non plus eu le temps de faire.

Et auriez-vous l'audace d'imprimer le code et de l'afficher dans votre salon ? (Ou plus prosaïquement, de le publier sur GitHub pour y faire participer la communauté ?)

## On a déjà « fait Java », pourquoi ce cours

Lavraie vie

```
graph TD; JDK[JDK] --> MonLogiciel[Mon Logiciel]; JDK --> ClientTiers[Client tiers]; MonLogiciel --> ClientTiers;
```

Mais souvent, dans la vraie vie<sup>6</sup>, on fait plutôt ça : on programme une bibliothèque réutilisable par des tiers.

6. mais rarement en 12...

## À propos de ce cours

Objectif : explorer les concepts de la programmation orientée objet (POO) au travers du langage Java et enseigner les principes d'une programmation fiable, pérenne et évolutive.

Contexte :

- Ce cours fait suite au cours de POO-IG de L2.
- Il précède les cours LOA de M1 (langage C++) et POCA de M2 (langage Scala).

Contenu :

- quelques révisions<sup>9</sup>, avec approfondissement sur certains thèmes,
- théme supplémentaire : la programmation concurrent (threads et APIs les utilisant)
- surtout, nous insisterons sur la POO (objectifs, principes, techniques, stratégies et patrons de conception, bonnes pratiques ...).

9. Trop à mon goût, mais elles sont nécessaires au moins pour s'assurer d'une terminologie commune. Si vous en voulez encore plus, relisez vos notes de l'année dernière !

## À propos de ce cours

### Organisation du cours

### Modalités de contrôle

#### Pourquoi Java ?

- Java convient tout à fait pour illustrer les concepts OO.
- (n + 1)ème contact avec Java → économie du ré-apprentissage d'une syntaxe → il reste du temps pour parler de POO.
- C'est aussi l'occasion de perfectionner la maîtrise du langage Java (habituellement, il faut plusieurs "couches")
- Et Java reste encore très pertinent dans le « monde réel ».
- Cela dit, d'autres langages<sup>12</sup> illustrent aussi ce cours (C, C++, OCaml, Scala, Kotlin, ...).

10. On pourra disséquer sur le côté langage OO "impur" de Java... mais Java fait l'affaire!

11. Les autres langages OO pour la JVM conviendraient aussi, mais sont moins connus.

12. Eux aussi installés en salles de TP. Soyez curieux, expérimentez !

#### Pourquoi Java ?

#### Volume horaire :

- 2h de CM toutes les 2 semaines,
  - 2h de TP chaque semaine.
- En ligne :** 1 cours sur Moodle pour télécharger tous les documents (ce cours, les fichiers de TP,...) et déposer vos travaux (dont les projets).  
→ code **CP005-2019**
- Inscrivez-vous vite !**
- Assurez-vous d'avoir vos identifiants pour vos comptes à l'UFR d'informatique en arrivant au premier TP !<sup>13</sup>**

13. Vous avez dû les obtenir à l'amphi de rentrée. Sinon, récupérez les auprès des administrateurs du réseau de l'UFR (batiment Sophie Germain, bureau 306).

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces graphiques

#### Gestion des erreurs et exceptions

#### Compléments en P00

#### Audrey Degorre

#### Introduction

#### Qu'est-ce que

#### Généralités

#### Style

#### Objets et classes

#### Types et polymorphisme

#### Héritage

#### Généricité

#### Concurrency

#### Interfaces

**Synthèse/résumé/à retenir**

Les pages de cette couleur contiennent une synthèse des dernières pages, c'est-à-dire soit un résumé, soit le résultat principal à retenir.

**Autres "codes"**

Autre que les codes de programmation, il existe d'autres types de codes :

- “Blah” : titre de paragraphe
- “important” : mot ou passage important
- “très important” : mot ou passage très important
- “concept” : concept clé (en général défini explicitement ou implicitement dans le transparent... sinon, il faudra s’assurer de connaître ou trouver la définition)
- “foreign words” : passage en langue étrangère
- “void” du code JavaEnLigne() {} : code Java intégré au texte

```
void duCodeJavaEnLigne() {
    System.out.println("Java_C'est cool!");
}
```

Code non intégré au texte.

**La programmation orientée objet**

Principes

- Principe de la POO : des messages 16 s'échangent entre des objets qui les traitent pour faire progresser le programme.
- **POO = paradigme centré sur la description de la communication entre objets.**
  - Pour faire communiquer un objet a avec un objet b, il est nécessaire et suffisant de connaître les messages que b accepte : l'**interface de b**.
  - Ainsi objets de même interface interchangables → **polymorphisme**.
  - Fonctionnement interne d'un objet 17 caché au monde extérieur → **encapsulation**.

Pour résumer : la POO permet de raisonner sur des **abstractions** des composants réutilisés, en ignorant leurs détails d'implémentation.

**Autres “codes”**

Autre que les codes de programmation, il existe d'autres types de codes :

- “Blah” : titre de paragraphe
- “important” : mot ou passage important
- “très important” : mot ou passage très important
- “concept” : concept clé (en général défini explicitement ou implicitement dans le transparent... sinon, il faudra s’assurer de connaître ou trouver la définition)
- “foreign words” : passage en langue étrangère
- “void” du code JavaEnLigne() {} : code Java intégré au texte

```
void duCodeJavaEnLigne() {
    System.out.println("Java_C'est cool!");
}
```

Code non intégré au texte.

<p><b>Discussion/ouverture</b></p> <p>Les pages de cette couleur contiennent une discussion, un commentaire, des remarques, ou bien une ouverture relative à ce qui vient d'être abordé.</p> <p>Ces pages doivent servir à faire réfléchir, mais ne sont en aucun cas à apprendre par cœur.</p>	<p><b>Autres "Codes"</b></p> <p>Références bibliographiques</p> <ul style="list-style-type: none"> <li>JLS : <i>Java language specification</i> (Oracle)</li> <li>JVMS : <i>Java virtual machine specification</i> (Oracle)</li> <li>EJ3 : <i>Effective Java 3rd edition</i> (Joshua Bloch)</li> <li>JCP : <i>Java Concurrency in Practice</i> (Brian Goetz)</li> <li>GOF : <i>Design Patterns : Elements of Reusable Object-Oriented Software</i> (Erich Gamma, Richard Helm, Ralph Johnson et John Vlissides a.k.a. "the Gang of Four")</li> <li>DPDP : <i>Design Principles and Design Patterns</i> (Robert C. Martin)</li> </ul> <p>(C'est juste un article, mais riche de principes utiles.)</p>	<p><b>La programmation orientée objet</b></p> <p>Avantages et raisons du succès de la POO</p> <ul style="list-style-type: none"> <li>POO permet de découper un programme en composants → code <b>robuste</b> et <b>évolutive</b> (composants testables et déboguables indépendamment et aisément remplaçables);</li> <li>réutilisables, au sein du même programme, mais aussi dans d'autres ; → facilite la création de logiciels de grande taille.</li> </ul> <p>POO → (discutable) façon de penser naturelle pour un cerveau humain "normal" 18 .</p> <p>18 Non "déformé" par des connaissances mathématiques pointues comme la théorie des catégories (cf. programmation fonctionnelle).</p>
---	---	---

## Approfondissement

en P20

Audrey Degorre

- Introduction
- Bases
- Généralités
- Style
- Objets et classes
- Types et polymorphisme
- Heritage
- Généricité
- Concurrency
- Interfaces graphiques
- Interfaces
- Objets et classes
- Types et polymorphisme
- Heritage
- Généricité
- Concurrency
- Interfaces
- Objets et classes
- Types et polymorphisme
- Heritage
- Généricité
- Concurrency

Supplément

Compléments en P20

Audrey Degorre

- Introduction
- Généralités
- Style
- Objets et classes
- Types et polymorphisme
- Heritage
- Généricité
- Concurrency
- Interfaces
- Objets et classes
- Types et polymorphisme
- Heritage
- Généricité
- Concurrency

Historique

## La programmation orientée objet

Audrey Degorre

- Paradigme de programmation inventé dans les années 1960 par Ole-Johan Dahl et Kristen Nygaard ([Simula](#) : langage pour simulation physique)...
- ... complété par Alan Kay dans les années 70 ([Smalltalk](#)) , qui a inventé l'expression "object oriented programming".
- Premiers langages OO "historiques" : [Simula 67](#) (1967<sup>14</sup>), [Smalltalk](#) (1980<sup>15</sup>).
- Autres LOO communs : C++, Objective-C, Eiffel, Java, Javascript, C#, Python, Ruby...

14. Simula est plus ancien (1952), mais il a intégré la notion de classe en 1967.

15. Première version publique de Smalltalk, mais le développement a commencé en 1971.

Java

Historique

- 1992 : langage Oak chez Sun [Microsystems](#)<sup>19</sup>, dont le nom deviendra Java ;
- 1996 : JDK 1.0 (première version de Java) ;
- 2009 : rachat de Sun par la société Oracle ;

En 2019, Java est donc « dans la force de l'âge » (23 ans)<sup>20</sup>, à comparer avec :

- même génération : Haskell : 29 ans, Python : 28 ans, JavaScript, OCaml : 23 ans
- anciens : C++ : 34 ans, C : 41 ans, COBOL : 60 ans, Lisp : 62 ans, Fortran : 65 ans...
- modernes : Scala : 15 ans, Go : 10 ans, Rust : 9 ans, Swift : 5 ans, Kotlin : 3 ans, ...

Depuis plusieurs années, Java est le 1<sup>er</sup> ou 2<sup>ème</sup> langage le plus populaire.<sup>21</sup>

19. Auteurs principaux : James Gosling et Patrick Naughton.

20. Ici, pour chaque langage, je prends en compte la version 1.0 ou équivalent.

21. Dans la plupart des classements principaux, TIOBE, RedMonk, PYPL. Selon les métriques, l'autre langage le plus populaire est C, Javascript ou Python.

**Versions « récentes » de Java :**

- **03/2014** : Java SE 8, la version long terme précédente,<sup>22</sup>
  - **09/2018** : Java SE 11, la version long terme actuelle,<sup>23</sup>
  - **03/2019** : Java SE 12, la dernière version
  - **17/09/2019** : Java SE 13, la prochaine version, imminente!
  - **09/2021** : Java SE 17, la prochaine version long terme.
- Ce cours utilise Java 11 (désormais installée en TP), mais :
- la plupart de ce qui est dit vaut aussi pour les versions antérieures;
  - les nouveautés de Java 12 et 13 ne sont pas censurées.
- <sup>22</sup>. Utilisée pour POCs et CP005 jusqu'à l'an passé.  
<sup>23</sup>. Depuis Java 9, une version "normale" sort tous les 6 mois et une à "support long terme", tous les 3 ans.

**Java****Compilateurs en PdO****Aide/Dégoûre****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****« Java » (Java SE) est en réalité une plateforme de programmation caractérisée par :**

- le langage de programmation Java
    - orienté objet à classes,
    - à la syntaxe inspirée de celle du langage C<sup>24</sup>,
    - à l'usage statique,
    - à gestion automatique de la mémoire, via son **ramasse-miettes** (*garbage collector*),
    - sa **machine virtuelle (JVM)**<sup>25</sup>, permettant aux programmes Java d'être multi-plateforme (le **code source** se compile en **code-octet** pour JVM, laquelle est implémentée pour nombreux types de machines physiques).
    - les bibliothèques officielles du JDK (fournissant l'**API**<sup>26</sup> Java), très nombreuses et bien documentées (+ nombreuses bibliothèques de tierces parties),
    - les bibliothèques standard de Java (fournies par l'environnement Java sans objet).
- <sup>24</sup>. C sans pointeurs et **struct**  $\simeq$  Java sans objet.
- <sup>25</sup>. Java Virtual Machine
- <sup>26</sup>. Application Programming interface

**Java****Compléments en PdO****Aide/Dégoûre****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction****Généralités****Java****Conventions et styles****Objets et classes****Types et polymorphisme****Héritage****Généricité****Concurrency****Interfaces****Gestion des erreurs et exceptions****Exécution****Révision****Java****Caractéristiques principales****Introduction**

## Le « programme » Java

### Question(s) de style

### Question de goût ?

#### Compléments en P00

Audrey Degorre

#### Architecte d'un programme compilé/distribuable

#### Compléments en P00

Audrey Degorre

##### Le code compilé :

- organisé de façon similaire au code source.
- Mais, à chaque un fichier `.java` correspond (au moins) un fichier `.class`.
- Un programme compilé est distribuable via une ou des archives `.jar`<sup>39</sup>.
- Si on utilise JFMS, il y a exactement un fichier `.jar` par module.
- Pour cela, ce qui est déjà écrit doit être **lisible et compréhensible**.
  - lisible par le programmeur d'origine
  - lisible par l'équipe qui travaille sur le projet
  - lisible par toute personne susceptible de travailler sur le code source (pour le logiciel libre : la Terre entière !)

39. C'est en réalité un fichier `.zip` avec quelques métadonnées supplémentaires.

40. Si un code source contient plus de commentaires que de code, c'est en réalité assez "touche".

#### Compléments en P00

Audrey Degorre

#### Nature grammaticale (1)

#### Nature grammaticale (2)

#### Concision versus information

#### Compléments en P00

Audrey Degorre

#### Introduction

#### Généralités

#### Style

#### Code

#### Interfacing

#### Généricité

#### Concurrency

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

#### Generique

#### Heritage

#### Interfaces

#### Graphiques

#### Gestion des erreurs et exceptions

## Nombre de caractères par ligne

Compléments en PdO

Audrey Degorre

Nombre de caractères par ligne	
• On limite le nombre de caractères par ligne de code. Raisons :	<ul style="list-style-type: none"><li>certaines personnes préfèrent désactiver le retour à la ligne automatique 51 ;</li><li>même la coupe automatique ne se fait pas forcément au meilleur endroit ;</li><li>longues lignes illisibles pour le cerveau humain (même si entièrement affichées) ;</li><li>certaines personnes aiment pouvoir afficher 2 fenêtres côté à côté.</li></ul>
• Limite traditionnelle : 70 caractères/ligne (les vieux terminaux ont 80 colonnes <sup>52</sup> ).	
De nos jours (écrans larges, haute résolution), 100-120 est plus raisonnable <sup>53</sup> .	
• Arguments contre des lignes trop courtes :	<ul style="list-style-type: none"><li>découpage trop élémentaire rendant illisible l'intention globale du programme ;</li><li>incitation à utiliser des identifiants plus courts pour pouvoir écrire ce qu'on veut en une ligne (→ identifiants peu informatifs, mauvaise pratique).</li></ul>
51. De plus, historiquement, les éditeurs de texte n'avaient pas le retour à la ligne automatique.	
52. Et d'où vient ce nombre 80 ? C'est le nombre de colonnes dans le standard de cartes perforées d'IBM inventé en... 1928 ! Et pourquoi ce choix en 1928 ? Parce que les machines à écrire avaient souvent 80 colonnes... bref c'est de l'histoire très ancienne !	
53. Selon moi, mais attention, c'est un sujet de débat houleux !	

## Indentation

Compléments en PdO

Audrey Degorre

Indentation	
• Indenter = mettre du blanc en tête de ligne pour souligner la structure du programme. Ce blanc est constitué d'un certain nombre d' <b>indentations</b> .	<ul style="list-style-type: none"><li>On essaye de privilier les retours à la ligne en des points du programme "hauts" dans l'arbre syntaxique (→ minimiser la taille de l'indentation).</li></ul>
En Java, typiquement, 1 indentation = 4 espaces (ou 1 tabulation).	<ul style="list-style-type: none"><li>P. ex., dans "<math>(x + 2) * (3 - 9 / 2)</math>", on préfèrera couper à côté de "*" → <math>(x + 2) * (3 - 9 / 2)</math></li></ul>
Le nombre d'indentations est égal à la profondeur syntaxique du début de la ligne	<ul style="list-style-type: none"><li>Le nombre de paires de symboles<sup>54</sup>, ouvertes mais pas encore fermées.<sup>55</sup></li></ul>
Tout éditeur raisonnablement évolué sait indenter automatiquement (règles paramétrables dans l'éditeur). Pensez à demander régulièrement l' <b>indentation automatique</b> , afin de vérifier qu'il n'y a pas d'erreur de structure !	<ul style="list-style-type: none"><li>Parfois difficile à concilier avec la limite de caractères par ligne → compromis nécessaires.</li><li>→ pour le lieu de coupure et le style d'indentation, essayez juste d'être raisonnable et consistant. Dans le cadre d'un projet en équipe, se référer aux directives du projet.</li></ul>

## Où couper les lignes

Compléments en PdO

Audrey Degorre

Où couper les lignes	
• On limite le nombre de caractères par ligne de code. Raisons :	<ul style="list-style-type: none"><li>certaines personnes préfèrent désactiver le retour à la ligne automatique 51 ;</li><li>même la coupe automatique ne se fait pas forcément au meilleur endroit ;</li><li>longues lignes illisibles pour le cerveau humain (même si entièrement affichées) ;</li><li>certaines personnes aiment pouvoir afficher 2 fenêtres côté à côté.</li></ul>
• Limite traditionnelle : 70 caractères/ligne (les vieux terminaux ont 80 colonnes <sup>52</sup> ).	
De nos jours (écrans larges, haute résolution), 100-120 est plus raisonnable <sup>53</sup> .	
• Arguments contre des lignes trop courtes :	<ul style="list-style-type: none"><li>découpage trop élémentaire rendant illisible l'intention globale du programme ;</li><li>incitation à utiliser des identifiants plus courts pour pouvoir écrire ce qu'on veut en une ligne (→ identifiants peu informatifs, mauvaise pratique).</li></ul>
51. De plus, historiquement, les éditeurs de texte n'avaient pas le retour à la ligne automatique.	
52. Et d'où vient ce nombre 80 ? C'est le nombre de colonnes dans le standard de cartes perforées d'IBM inventé en... 1928 ! Et pourquoi ce choix en 1928 ? Parce que les machines à écrire avaient souvent 80 colonnes... bref c'est de l'histoire très ancienne !	
53. Selon moi, mais attention, c'est un sujet de débat houleux !	

## Nombre de paramètres des méthodes

Compléments en PdO

Audrey Degorre

Nombre de paramètres des méthodes	
• Autre critère : le nombre de paramètres.	<ul style="list-style-type: none"><li>Autre critère : le nombre de paramètres.</li></ul>
• Trop de paramètres (>4) implique :	<ul style="list-style-type: none"><li>Trop de paramètres (&gt;4) implique :</li><li>Une signature longue et illisible.</li><li>Une utilisation difficile ("ah mais ce paramètre là, il était en 5e ou en 6e position, déjà ?")</li></ul>
• Il est souvent possible de réduire le nombre de paramètres	<ul style="list-style-type: none"><li>Il est souvent possible de réduire le nombre de paramètres</li><li>en utilisant la surcharge,</li><li>ou bien en séparant la méthode en plusieurs méthodes plus petites (en décomposant la tâche effectuée),</li><li>ou bien en passant des objets composites en paramètre</li></ul>
• Exemple :	<p>voici un exemple ( qui n'est pas du Java ; mais suit ses "conventions d'indentation"</p> <p>)</p> <p>54. Parenthèses, crochets, accolades, guillemets, chevrons, ...</p> <p>55. Pas seulement : les règles de priorité des opérations créent aussi de la profondeur syntaxique.</p>

## Taille des méthodes

Compléments en PdO

Audrey Degorre

Taille des méthodes	
• Pour une méthode, la taille est le nombre de lignes.	<ul style="list-style-type: none"><li>Pour une méthode, la taille est le nombre de lignes.</li></ul>
• Principe de responsabilité unique <sup>57</sup> : une méthode est censée effectuer une tâche précise et compréhensible.	<ul style="list-style-type: none"><li>Principe de responsabilité unique<sup>57</sup> : une méthode est censée effectuer une tâche précise et compréhensible.</li></ul>
• → Un excès de lignes	<ul style="list-style-type: none"><li>→ Un excès de lignes</li></ul>
• nuit à la compréhension ;	<ul style="list-style-type: none"><li>nuit à la compréhension ;</li></ul>
• peut traduire le fait que la méthode effectue en réalité plusieurs tâches probablement séparables.	<ul style="list-style-type: none"><li>peut traduire le fait que la méthode effectue en réalité plusieurs tâches probablement séparables.</li></ul>
• Quelle est la bonne longueur ?	<ul style="list-style-type: none"><li>Quelle est la bonne longueur ?</li></ul>
• Mon critère <sup>58</sup> : on ne peut pas bien comprendre une méthode si on ne peut pas la parcourir en un simple coup d'œil	<ul style="list-style-type: none"><li>Mon critère<sup>58</sup> : on ne peut pas bien comprendre une méthode si on ne peut pas la parcourir en un simple coup d'œil</li></ul>
• → faire en sorte qu'elle tienne en un écran (~ 30-40 lignes max.)	<ul style="list-style-type: none"><li>→ faire en sorte qu'elle tienne en un écran (~ 30-40 lignes max.)</li></ul>
• En général, suivre les directives du projet.	<ul style="list-style-type: none"><li>En général, suivre les directives du projet.</li></ul>
57. Oui, là aussi !	<p>57. Oui, là aussi !</p>
58. qui n'enjuge que moi !	<p>58. qui n'enjuge que moi !</p>

## Taille des classes

Compléments en PdO

Audrey Degorre

Taille des classes	
• Quelle est la bonne taille pour une classe ?	<p>Quelle est la bonne taille pour une classe ?</p>
• Déjà, plusieurs critères de taille : nombre de lignes, nombre de méthodes, ....	<ul style="list-style-type: none"><li>Déjà, plusieurs critères de taille : nombre de lignes, nombre de méthodes, ....</li></ul>
• Le découpage en classes est avant tout guidé par l'abstraction objet retenue pour modéliser le problème qu'on veut résoudre.	<ul style="list-style-type: none"><li>Le découpage en classes est avant tout guidé par l'abstraction objet retenue pour modéliser le problème qu'on veut résoudre.</li></ul>
• En pratique, une classe trop longue est désagréable à utiliser. Ce désagréement traduit souvent une décomposition insuffisante de l'abstraction. <sup>59</sup>	<ul style="list-style-type: none"><li>En pratique, une classe trop longue est désagréable à utiliser. Ce désagréement traduit souvent une décomposition insuffisante de l'abstraction.<sup>59</sup></li></ul>
• Conseil : se fixer une limite de taille et décider, au cas par cas, si et comment il faut "réparer" les classes qui dépassent la limite (cela incite à améliorer l'aspect objet du programme).	<ul style="list-style-type: none"><li>Conseil : se fixer une limite de taille et décider, au cas par cas, si et comment il faut "réparer" les classes qui dépassent la limite (cela incite à améliorer l'aspect objet du programme).</li></ul>
• En général, pour un projet en équipe, suivre les directives du projet.	<ul style="list-style-type: none"><li>En général, pour un projet en équipe, suivre les directives du projet.</li></ul>
56. Le « S » de « SOLID » : single responsibility/principe/principe de responsabilité unique.	

## Commentaires

Compléments en PdO

Audrey Degorre

Commentaires	
• Plieurs sortes de commentaires (1)	<p>Plieurs sortes de commentaires (1)</p>
• En ligne :	<p>En ligne :</p> <pre><code>int length; // Length of this or that</code></pre>
• Pratique pour un commentaire très court tenant sur une seule ligne (ou ce qu'il en reste...)	<p>Pratique pour un commentaire très court tenant sur une seule ligne (ou ce qu'il en reste...)</p>
• en bloc :	<p>en bloc :</p> <pre><code>/* Un commentaire un peu plus long.  * Les /* intermédiaires ne sont pas obligatoires, mais Eclipse  * les ajoute automatiquement pour le style. Laissez-les !  */</code></pre>
• À utiliser quand vous avez besoin d'écrire des explications un peu longues, mais que vous ne souhaitez pas voir apparaître dans la documentation à proprement parler (la JavaDoc).	<p>À utiliser quand vous avez besoin d'écrire des explications un peu longues, mais que vous ne souhaitez pas voir apparaître dans la documentation à proprement parler (la JavaDoc).</p>

## Patrons de conception (1) ou design patterns

Introduction

- Généralités
- Style
- Types et classes
- Heritage
- Generalité
- Concurrence
- Interfaces graphiques
- Gestion des erreurs et exceptions

Compléments en P00

Audrey Degorre

## Objets

Compléments en P00

Audrey Degorre

Introduction

- Généralités
- Style
- Types et classes
- Heritage
- Generalité
- Concurrence
- Interfaces graphiques
- Gestion des erreurs et exceptions

Discussion

Analogie langage naturel : patron de conception = figure de style

- Ce sont des stratégies standardisées et éprouvées pour arriver à une fin.
- ex : créer des objets, décrire un comportement ou structurer un programme
- Les utiliser permet d'éviter les erreurs les plus courantes (pour peu qu'on utilise le bon patron !) et de rendre ses intentions plus claires pour les autres programmeurs qui connaissent les patrons employés.
- Connaitre les noms des patrons permet d'en discuter avec d'autres programmeurs.<sup>59</sup>

59. De la même façon qu'apprendre les figures de style en cours de Français, permet de discuter avec d'autres personnes de la structure d'un texte....

Object = entité...

- caractérisée par un enregistrement contigu de données typées (**attributs**)<sup>62)</sup>
- accessible via une référence<sup>63</sup> vers cet enregistrement;
- manipulable/interrogeable via un ensemble de **méthodes** qui lui est propre.

classe de l'objet:	réf → Personne; <b>class</b>
int	age 42
String nom	réf → chaîne "Dupont"
String prenom	réf → chaîne "Toto"
...	
<b>boolean</b> marié	<b>true</b>

La variable référence      référence      l'objet  
**Personne** **toTo**

62. On dit aussi champs comme pour les **struct** de C/C++.

Pour la représentation mématoire, un objet et une instance de **struct** sont similaires.

63. Il s'agit en vrai d'un pointeur. Les références de C++ sont un concept (un peu) différent.

Introduction

- Généralités
- Style
- Types et classes
- Heritage
- Generalité
- Concurrence
- Interfaces graphiques
- Gestion des erreurs et exceptions

Compléments en P00

Audrey Degorre

## Objets

Autre vision bas niveau avec le haut niveau

Compléments en P00

Audrey Degorre

Introduction

- Généralités
- Style
- Types et classes
- Heritage
- Generalité
- Concurrence
- Interfaces graphiques
- Gestion des erreurs et exceptions

Discussion

Question : où arrêter le graphe d'un objet ?

- Est-ce que les éléments d'une liste font partie de l'objet-liste?
- En exagérant un peu, un programme ne contient en réalité qu'un seul objet!<sup>66</sup>
- Clairement, le graphe d'un objet ne doit pas contenir tous les enregistrements accessibles depuis l'enregistrement principal. Mais où s'arrêter et sur quel critère ?

Cela n'est pas anodin :

- Que veut dire « copier » un objet? (Quelle « profondeur » pour la copie?)
- S'il parle d'un objet non modifiable, qu'est-ce qui n'est pas modifiable?
- Est-ce qu'une collection non modifiable peut contenir des éléments modifiables?

Cette discussion a trait aux notions d'encapsulation et de composition. À suivre!

66. En effet : les enregistrements non référencés par le programme, sont assez vite détruits par le GC.

Discussion

<h2 style="color: #0070C0;">Patrons de conception (2)</h2> <p style="font-size: small; transform: rotate(-90deg);">ou design pattern</p>	<ul style="list-style-type: none"> <li>● Quelques exemples dans le cours : décorateur, délégation, observateur/observable, moniteur.</li> <li>● Patrons les plus connus décrits dans le livre du "Gang of Four" (GoF) 60             <ul style="list-style-type: none"> <li>• les patrons ne sont pas les mêmes d'un langage de programmation à l'autre :</li> <li>• les patrons implémentables dépendent de ce que la syntaxe permet</li> <li>• les patrons utiles dépendent aussi de ce que la syntaxe permet :</li> <li>• quand un langage est créé, sa syntaxe permet souvent de traiter simplement des situations qui autrefois nécessitaient l'usage d'un patron (moins simple).</li> </ul> </li> <li>● Probablement, le concept de déclarer des classes est apparu comme cela.</li> </ul>
<p style="background-color: #0070C0; color: white; padding: 10px; text-align: center;"><b>Objet</b></p> <p style="font-size: small;">Quelle bonne vision bas niveau</p> <p>Cette dernière vision est en fait réductrice. En POO, on veut <b>s'abstraire</b> de l'implémentation concrète des concepts.</p> <p>À service égal, les objets Personne peuvent aussi être représentés ainsi :</p> <pre style="border: 1px solid black; padding: 10px; margin-left: 20px;">     classe : ↗ Personne.class     int age 42 →     Ident ident → ...     boolean mariage true →     String prenom → Toto     String nom → "Dupont"   </pre> <p>Les méthodes seraient écrites différemment mais, à l'usage, cela ne se verrait pas.<sup>64</sup></p> <p>Pourtant cela aurait encore du sens de parler d'objets Personne contenant les propriétés nom et prenom.</p> <p>64. À condition qu'on n'utilise pas directement les attributs. D'où l'intérêt de les rendre privés !</p> <p style="background-color: #0070C0; color: white; padding: 10px; text-align: center;"><b>Classe</b></p> <p style="font-size: small;">Langages à prototypes, langages à classes</p> <ul style="list-style-type: none"> <li>● Besoin : créer de nombreux objets similaires (même interface, même schéma de données).</li> <li>● 2 solutions → 2 familles de LOO :             <ul style="list-style-type: none"> <li>• LOO à classes (java et la plupart des LOO) : les objets sont instanciés à partir de la description donnée par une <b>classe</b>.</li> <li>• LOO à prototypes (toutes les variantes d'ECMAScript dont JavaScript ; Self, Lissac, ...) : les objets sont obtenus par extension d'un objet existant (le prototype).</li> </ul> </li> </ul> <p style="text-align: right;">→ l'existence de classes n'est pas une nécessité en POO</p>	

Classes

Points de vue pertinents pour Java

Classes	Exemple (le même en UML)
<p>Style</p> <p>Types et polymorphisme</p> <p>Heritage</p> <p>Géométrie</p> <p>Concurrente</p> <p>Interfaces graphiques</p> <p>Gestion des erreurs et exceptions</p>	<pre> <b>Personne</b>  - nom : String - age : int - marié : boolean  + &lt;&lt; Create &gt;&gt; Personne(nom : String, age : int, marié : boolean) : Personne + getName() : String + setName(nom : String) + getAge() : int + setAge(age : int) + getMarried() : boolean + setMarried(marie : boolean) </pre>

Pour l'objet juste donné en exemple, la classe Personne pourrait être :

```

public class Personne {
    // attributs
    private String nom; private int age; private boolean marie;

    // constructeur
    public Personne(String nom, int age, boolean marie) {
        this.nom = nom; this.age = age; this.marie = marie;
    }

    // méthodes (ici : accessseurs)
    public String getNom() { return nom; }
    public void setNom(String nom) { this.nom = nom; }

    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }

    public boolean getMarie() { return marie; }
    public void setMarie(boolean marie) { this.marie = marie; }
}

```

- Une classe permet de "fabriquer" plusieurs objets selon un même modèle : les **instances**<sup>70</sup> de la classe.
- Ces objets ont le même type, dont le nom est celui de la classe.
- La fabrication d'un objet s'appelle l'**instanciation**. Celle-ci consiste à
  - réservé la mémoire ( $\sim$  `malloc` en C)
  - initialiser les données<sup>71</sup> de l'objet
- On instancie la classe `Truc` via l'expression "`new Truc(params)`", dont la valeur est une référence vers un objet de type `Truc` nouvellement créé.<sup>72</sup>

Classe =	Autres points de vue (non-OO) :
• sous-division <u>yntaxique</u> du programme	Autres objets et éléments de programmation
• espace de noms (définitions de nom identique possibles si dans classes différentes)	Objets et éléments de programmation
• parfois, juste une bibliothèque de fonctions statiques, non instanciable <sup>69</sup>	Heritage
• exemples de classes non instanciables du JDK : <b>System</b> , <b>Arrays</b> , <b>Math</b> , ...	Généricité
Les aspects ci-dessus sont pertinents en Java, mais ne retenir que ceux-ci serait manquer l'essentiel : i.e. : <b>classe</b> = <b>concept de POO</b> .	Concurrence interfaces générales Gestion des erreurs et exceptions
	Révision

Membres d'une classe	Le <b>corps</b> d'une classe C consiste en une séquence de définitions : constructeurs <sup>77</sup> et membres <sup>78</sup> de la classe.
Plusieurs catégories de membres : attributs, méthodes et types membres <sup>78</sup> .	<p>Un membre m peut être</p> <ul style="list-style-type: none"><li>soit non statique ou <b>d'instance</b> (relatif à une instance de C)</li><li>Utilisable en écrivant « m » n'importe où ou un <b>this</b> (<b>récepteur implicite</b>) de type C existe et ailleurs en écrivant « <b>récepteurDeTypeC.m</b> ».</li><li>soit <b>statique</b> (relatif à la classe C) → mot-clé <b>static</b> dans déclaration.</li><li>Utilisable sans préfixe dans le corps de C et ailleurs en écrivant « <b>C.m</b> ».</li></ul> <p>Les <b>membres d'un objet</b> donné sont les membres non statiques de la classe de l'objet.</p> <p>77. D'après la JLS 8.2, les constructeurs ne sont pas des membres. Néanmoins, sont déclarés à l'intérieur d'une classe et acceptent, comme les membres, les modificateurs de visibilité (<b>private</b>, <b>public</b>...).</p> <p>78. Souvent abusivement appelées « classes internes ».</p>

Instanciation	Constructeurs (2)
	Introduction Généralités
	Style Système et classes Tutoriel classe Nomination et construction Type retourneur Types et paramètres Héritage Généralité Concurrente Interfaces graphiques Gestion des erreurs et exceptions Révision
<b>Il est possible de :</b>	<ul style="list-style-type: none"> <li>• définir plusieurs constructeurs (tous le même nom → cf. <u>surcharge</u>);</li> <li>• définir un constructeur secondaire à l'aide d'un autre constructeur déjà défini : <u>sa première instruction doit alors être <b>this</b>(paramètres) au réconstructeur</u> ;</li> <li>• ne pas écrire le constructeur :           <ul style="list-style-type: none"> <li>• Si on ne le fait pas, le compilateur ajoute un <b>constructeur par défaut</b> sans paramètre.</li> <li>• Si on a écrit un constructeur, alors il n'y a pas de constructeur par défaut.</li> </ul> </li> </ul>
74. Ou bien <b>super</b> (paramètres) : si utilisation d'un constructeur de la superclasse.	
75. Les attributs restent à leur valeur par défaut (0, <b>false</b> ou <b>null</b> ), ou bien à celle donnée par leur initialisateur, si y en a un.	
76. Mais rien n'empêche d'écrire en plus, à la main, un constructeur sans paramètre.	



## Encapsulation

En passant : niveaux de visibilité pour les déclarations de premier niveau

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Au contraire de nombreux autres principes exposés dans ce cours, l'**encapsulation ne favorise pas directement la réutilisation de code.**
- À première vue, c'est le contraire : on interdit l'utilisation directe de certaines parties de la classe.
- En réalité, l'encapsulation augmente la confiance dans le code réutilisé (ce qui, indirectement, peut inciter à le réutiliser davantage).

### Exemple :

```
class A {
    int x; // visible dans le package
    private double ; // visible seulement dans A
    public final String nom = "Toto"; // visible partout
}
```

90. voir héritage

Discussion

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

L'encapsulation est mise en œuvre via les **modificateurs de visibilité** des membres.

- 4 niveaux de visibilité en faisant précéder leur déclaration de **private**, **protected** ou **public** ou d'aucun de ces mots (→ visibilité **package-private**).

Visibilité	classe	paquetage	sous classes	partout
<b>private</b>	X			
<b>package-private</b>	X	X		
<b>protected</b>	X	X	X	
<b>public</b>	X	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

### 91. Précisions/rappels :

- 'premier niveau' = hors des classes, directement dans le fichier;
- seules les déclarations de type (classes, interfaces, enumérations, annotations) sont concernées.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

## Encapsulation

Compléments en P00  
Aidez Degorre

Notion de visibilité : s'applique aussi aux déclarations de premier niveau<sup>91</sup>.

- Ici, 2 niveaux seulement : **public** ou **package-private**.

Visibilité	package-private	paquetage	partout
<b>private</b>	X		
<b>package-private</b>	X	X	
<b>protected</b>	X	X	
<b>public</b>	X	X	X

- Rappel** : une seule déclaration publique de premier niveau autorisée par fichier. La classe/interface... définit porte alors le même nom que le fichier.

## Encapsulation

Encapsulation	Accesseurs (get, set, ...) et propriétés (4)
	Introduction
	Généralités
	Style
	obligatoires
	classiques
	classiques
	modernes
	modernes
	Types
	paramétrés
	Héritage
	Généralisation
	Couplage
	Interfaçage
	Gestionnaire
	exception
<b>Exemple :</b> Propriété en lecture seule avec évaluation paresseuse.	<pre>public final class Entier {     public Entier(int valeur) { this.valeur = valeur; }      private final int valeur;      // propriété "diviseurs"     private List&lt;Integer&gt; diviseurs;      public List&lt;Integer&gt; getDiviseurs() {         if (diviseurs == null) diviseurs =             Collections.unmodifiableList(Outils.factorise(valeur)); // &lt;- calcul         // coûteux, à n effectuer que si nécessaire         return diviseurs;     } }</pre>

<b>Compléments</b> en P00 Astrée Bégorre	<b>Encapsulation</b> Accessseurs (get, set, ...) et propriétés (5)	<p><b>Introduction</b></p> <p><b>Généralités</b></p> <p><b>Sytle</b></p> <p><b>Objet et classes</b></p> <p><b>Attributs et constructeurs</b></p> <p><b>Initialisation</b></p> <p><b>Exemples</b></p> <p><b>Types et polymorphisme</b></p> <p><b>Héritage</b></p> <p><b>Généricité</b></p> <p><b>Concurrente</b></p> <p><b>Interfaces graphiques</b></p> <p><b>Gestion des erreurs et exceptions</b></p>	<p><b>Comportements envisageables pour get et set :</b></p> <ul style="list-style-type: none"> <li>• contrôle de validité avant modification;</li> <li>• initialisation par défaut : la valeur de la propriété n'est calculée que lors du premier accès (et non dès la construction de l'objet);</li> <li>• consignation dans un journal pour débogage ou surveillance;</li> <li>• observabilité : le seteur notifie les objets observateurs lors des modifications;</li> <li>• vétérabilité : le seteur n'a d'effet que si aucun objet (dans une liste connue de "vét-eurs") ne s'oppose au changement;</li> </ul>
--	---	---	---

<p><b>Encapsulation</b></p> <p>Aliasинг : pourquoi les restrictions de visibilité ne suffisent pas pour garantir l'encapsulation</p>	<p><b>Aliasing</b> = existence de références multiples vers un même objet.</p>	 <pre> graph LR     ref1[ref1] --&gt; objet[objet]     ref2[ref2] --&gt; objet     </pre>	<p>Quand un attribut référence un objet qui est aussi référencé à l'extérieur de cette classe, le bénéfice de l'encapsulation est alors annulé.</p>	<p>Attribut privé de C →  → reference externe à C</p>	<p>À éviter :</p> <p>Cela revient<sup>105</sup> à laisser l'attribut en public puisque le détenteur de cette référence peut faire les mêmes manipulations sur cet objet que la classe contenant l'attribut.</p>	<p>105. Quasiment : en effet, si l'attribut est privié, il reste impossible de modifier la valeur de l'attribut, i.e. l'autresse qu'il stocke, depuis l'extérieur.</p>
<p>Compléments en P2O</p>	<p>Introduction Généralités Style Objets et classes Thème et contexte Mécanismes et règles</p>	<p>Fonctionnement Types et polymorphisme Héritage Généricité Concurrente</p>	<p>Interfaces Informatique générale Gestion des erreurs et exceptions</p>			<p>Discussion</p>

Attribut	Initialisation	Type	Objet	Classe	Interfaz	Entidad
A	new Data(x)	Data	d	Client	use(Data d)	Client
B	x = y	int	d	Client	use(Data d)	Client
C	this = x	Data	d	Client	use(Data d)	Client
D	new Data(x)	Data	d	Client	use(Data d)	Client

Lesquelles des classes A, B, C et D garantissent que l'entier contenu dans l'attribut d garde la valeur qu'on y a mise à la construction ou lors du dernier appel à setData ?

```

class Data {
    int x;
    public Data(int x) { this.x = x; }
    public Data copy() { return new Data(x); }
}

class Client {
    private final Data d;
    public Client(Data d) { this.d = d; }

    class Data {
        private final Data d;
        public void use(Data d) { this.d = d.copy(); }
        public void use(Data d) { Client.use(d); }
        public Data getDatal() { return d; }
    }
}

```

Reviens à répondre à : les attributs de ces classes peuvent-ils avoir des alias extérieurs ?

<p><i>Aliasing</i> : comment l'émpêcher.</p>	<p><b>Aliasing souvent indésirable (<u>pas toujours !</u>) → il faut savoir l'empêcher. Pour cela :</b></p> <pre data-bbox="687 831 957 1372"> <b>class A {</b>     // Mettre les attributs sensibles en private :     <b>private Data data;</b>     // Et effrayer des copies détéfensives (E3 Item 50)...     // - de tout objet qu'on souhaite partager,     // - qu'il soit retourné par un getteur :     <b>public Data getdata() { return data.copy(); }</b>     // ou passé en paramètre d'une méthode extérieure :     <b>public void doSomething (X bar(Data data).copy());</b>     // - de tout objet passé en argument pour être stocké dans un attribut     // - que soit dans les méthodes     <b>public void setdata(Data data) { this.data = data.copy(); }</b>     // - ou dans les constructeurs     <b>public A (Data data) { this.data = data.copy(); } //</b> } </pre> <p>Résumé : <u>ni diviser</u> ses références, <u>ni conserver</u> une référence qui nous a été donnée.</p>
--	---

Qu'est-ce que c'est et comment la réalise-t-on ?

- **Copie défensive** = copie profonde réalisée pour éviter des flas indésirables.
- **Copie profonde** : technique consistant à obtenir une copie d'un objet « égale » à son original au moment de la copie, mais dont les évolutions futures seront indépendantes.
- **2 cas, en fonction du genre de valeur à copier :**
  - Si type primaire ou immuable [107] : pas d'évolutions futures → une copie directe suffit.
  - Si type mutable → on crée un nouvel objet dont les attributs contiennent des copies profondes des attributs de l'original (et ainsi de suite, récursevement : on copie le graphique de l'objet [108]).
- 106. La relation d'égalité est celle donnée par la méthode equal [5].
- 107. Type **immuable** (*immutable*) : type (en fait toujours une classe), dont toutes les instances sont des objets non modifiables.
- C'est une propriété souvent recherchée, notamment en programmation concurrente.
- Contre : **mutable** (*mutable*)
- 108. Isavoir en quoi consiste le graphique de l'objet, sinon la notion de copie profonde reste ambiguë.

<h2>Copie défensive</h2> <p>Qu'est-ce que c'est et comment la réalise-t-on ? (exemple)</p>	<pre>public class Item {     int productNumber; Point location; String name;     public Item copy() { // Item est mutable donc cette méthode est utile         Item ret = new Item();         ret.productNumber = productNumber; // int est primitif, une copie simple suffit         ret.location = new Point(location.x, location.y); // Point est mutable, il faut         // une copie profonde         ret.name = name; // String est immuable, une copie simple suffit         return ret;     } }</pre>	<p><b>Remarque :</b> il est impossible<sup>109</sup> de faire une copie profonde d'une classe mutable dont on n'a pas l'autorité si ses attributs sont privés et l'auteur n'a pas prévu lui-même la copie.</p>
		<small>109. Sauf à utiliser la réflexion... mais dans le cadre du JPM, il ne faut pas trop compter sur celle-ci.</small>

<sup>109.</sup> Sauf à utiliser la réflexion... mais dans le cadre du JPMs, il ne faut pas trop compter sur celle-ci.



<p>Soit <b>TI</b> un type imbriqué dans <b>TE</b>, type englobant. Alors, dans <b>TI</b> :</p> <ul style="list-style-type: none"> <li>• <b>this</b> désigne toujours (quand elle existe) l'instance courante de <b>TI</b>;</li> <li>• <b>TE.this</b> désigne toujours (quand elle existe) l'<b>instance englobante</b>, c.-à-d. l'instance courante de <b>TE</b>, c.-à-d. :</li> </ul> <ul style="list-style-type: none"> <li>• si <b>TI</b> classe membre non statique, la valeur de <b>this</b> dans le contexte où l'instance courante de <b>TI</b> a été créée. Exemple :</li> </ul>	<pre>class CE {     int x = 1; } class CT {     int y = 2;     void f() {         System.out.println(CE.this.x + " " + this.y);     } }  // alors new (CE().new CT()).f(); affichera "1 2"</pre>
	<ul style="list-style-type: none"> <li>• La référence <b>TE.this</b> est en fait stockée dans toute instance de <b>TI</b> (attribut caché).</li> </ul>

<p>Complexe en per- sistance</p> <p>Audier des événements</p> <p>Introduction</p> <p>Généralités</p> <p>Style</p> <p>Opérations et classes</p> <p>Classes et interfaces</p> <p>Observer et observer</p> <p>Observer et observer</p> <p>Observateurs</p> <p>Types et polymorphisme</p> <p>Héritage</p> <p>Généricité</p> <p>Généricité</p> <p>Interfacing</p> <p>Gestion des erreurs et exceptions</p> <p>Exemples</p>	<p>Exemple de classe locale</p> <p><b>Types imbriqués</b></p> <p><b>La définition de classe se place comme une instruction dans un bloc (gén. une méthode) :</b></p> <pre>class Maliste&lt; implements List&lt;&gt; {     ...     public Iterator&lt;&gt; iterator() {         class MonIterator implements Iterator&lt;&gt; {             public boolean hasNext() { ... }             public Object next() { ... }             public void remove() { ... }         }         return new MonIterator();     } }</pre> <p><b>En plus des membres du type englobant, accès aux autres déclarations du bloc (notamment variables locales [23]).</b></p>
---	--

**Types imbriqués**

Exemple de classe anonyme

---

**La définition de classe est une expression dont la valeur est une instance<sup>124</sup> de la classe**

Generalités

Chaque et  
chaque classe  
possède un  
constructeur  
privé nommé  
`__construct()`.

Autre exemple :

```
class Liste {>
    implements List<T> {
        ...
        public Iterator<T> iterator() {
            return /* de là */ /> new Iterator<T>() {
                ...
                public boolean hasNext() {
                    public T next() {
                        ...
                    }
                }
                ...
            } /* à là */
        }
    }
}
```

---

**124. La seule instance.**

Complements  
en POO

Introduction

Generalités

Style

Classes et  
classes  
internes

Interfaces

Type envoi

Types et  
Héritage

Généricité

Concurrency

Interfaces  
graphiques

Gestion des  
erreurs et  
exceptions

Exemple

<p><b>Classes anonymes</b> =</p> <ul style="list-style-type: none"> <li>cas particulier de classe locale avec <u>yntaxe allégée</u> ; → comme classes locales, accès aux déclarations du bloc<sup>125</sup> ;</li> <li>déclaration "en ligne" : c'est syntaxiquement une <u>expression</u>, qui s'évalue comme une instance de la classe déclarée ;</li> <li>déclaration de classe sans donner de nom ⇒ instanciable une seule fois → c'est une classe singleton ;</li> <li>autre restriction : un seul supertype direct<sup>126</sup> (dans l'exemple : <u>Iterator</u>) .</li> </ul> <p>Question : comment exécuter des instructions à linitialisation d'une classe anonyme alors qu'il n'y a pas de constructeur ? → Réponse : utiliser un "Bloc dinitialisation" (Au besoin, recherchez ce que c'est.)</p>	<p><b>Syntaxe encore plus concise : lambda-expressions</b> (cf. chapitre dédié), par ex.</p> <p>x → <u>System.out.println(x)</u>.</p> <p>125. Avec la même pertinence : variables locales effectivement finales seulement.</p> <p>126. Une classe peut néanmoins, sauf dans ce cas, implémenter de multiples interfaces.</p>
--	--

Types imbriqués	Accessibilité au contexte englobant
<pre>class/interface/enum TypeEnglobant {     static int x = 4;     static class/interface/enum TypeMembre { static int y = x; }     static int z = typeMembre.y;</pre>	<p>Le contexte interne du type imbriqué contient toutes les définitions du contexte externe.</p> <p>Ainsi, sont accessibles directement (sans chemin<sup>128</sup>) :</p> <ul style="list-style-type: none"> <li>• dans tous les cas : les membres statiques du type englobant;</li> <li>• pour les classes membres non statiques et classes locales dans bloc non statique <u>tous</u> les membres non statiques du type englobant;</li> <li>• pour les classes locales : les définitions locales<sup>129</sup>.</li> </ul> <p>Réciproquement, depuis l'<u>extérieur</u> de <b>T1</b>, accès au membre <b>y</b> de <b>T1</b> en écrivant <b>T1.y</b>:</p> <p>128. sauf si l'<u>extérieur</u> de <b>T1</b>, accès au membre <b>y</b> de <b>T1</b> en écrivant <b>T1.y</b>.</p> <p>129. seulement effectivement finalisées pour les variables...</p>

**Plusieurs sortes de types imbriqués**

D'accord, mais lequel choisir ? [31]

```

graph TD
    A[Utilisé dans plusieurs méthodes?] -- non --> B[Utilisé dans plusieurs types niveau package?]
    A -- oui --> C[Instance dans plusieurs fois ou bien besoin de plusieurs types parent?]
    B -- non --> D[classe anonyme]
    B -- oui --> E[classe locale]
    C -- non --> F[classe membre non statique]
    C -- oui --> G[classe membre statique]
    E -- non --> H[Instances dépendant d'une instance englobante?]
    E -- oui --> I[type membre statique]
    F -- non --> J[type membre package]
    F -- oui --> K[Instances dépendant d'une instance englobante?]
    G -- non --> L[type membre package]
    G -- oui --> M[Instances dépendant d'une instance englobante?]
    H -- non --> N[C'est à dire non imbriqués, définis directement dans les fichiers .java.]
    H -- oui --> O[Effectif Java 3rd édition, item 24 : Favor static member classes over nonstatic.]
    I -- non --> P[Effectif Java 3rd édition, item 24 : Favor static member classes over nonstatic.]
    I -- oui --> Q[C'est à dire non imbriqués, définis directement dans les fichiers .java.]
    K -- non --> R[Effectif Java 3rd édition, item 24 : Favor static member classes over nonstatic.]
    K -- oui --> S[C'est à dire non imbriqués, définis directement dans les fichiers .java.]
    L -- non --> T[Effectif Java 3rd édition, item 24 : Favor static member classes over nonstatic.]
    L -- oui --> U[C'est à dire non imbriqués, définis directement dans les fichiers .java.]
    M -- non --> V[Effectif Java 3rd édition, item 24 : Favor static member classes over nonstatic.]
    M -- oui --> W[C'est à dire non imbriqués, définis directement dans les fichiers .java.]
  
```

- Types de données en POO
  - Introduction
  - Objectif et classes
  - Types et polymorphisme
  - Le principe des types
  - Basé sur l'héritage
  - Polymorphisme
  - Sous-type
  - Supertypes
  - Inheritance
- Introduction
- Objectives and classes
- Types and polymorphism
- The principle of types
- Based on inheritance
- Polymorphism
- Subtypes
- Supertypes
- Inheritance

On parle de différents **systèmes de types**.



Stades de vérification et statique	
Conférences en POG Autric Programe	<h2>Type dynamique et statique</h2> <p>quand vérifier la cohérence des données ?</p> <ul style="list-style-type: none"> <li>La vérification du bon typage d'un programme peut avoir lieu à différents moments :             <ul style="list-style-type: none"> <li>langages très « bas niveau » (assemblage x86, p. ex.) : jamais ;</li> <li>C, C++, OCaml... : dès la compilation (<b>type statique</b>) ;</li> <li>Python, PHP, Javascript, ... : seulement à l'exécution (<b>type dynamique</b>) ;</li> </ul> </li> </ul> <p>Remarque : types statique et dynamique ne sont pas mutuellement exclusifs. 147</p> <p>Les entités auxielles ont attribué un type ne sont pas les mêmes selon le moment où cette vérification est faite.</p> <p>Type statique → concerne les expressions du programme</p> <p>Type dynamique → concerne les données existant à l'exécution.</p> <p>Où se Java se situe-t-il ? Que type-t-on en Java ?</p> <p>148. Il existe même des langages où le programmeur décide ce qui est vérifié à l'exécution ou à la compilation. « type gradiel »,</p> <p>149. morceaux de texte évaluables dans le programme</p> <p>150. Ces entités n'existent pas avant l'exécution, de toute façon !</p>
Compléments en POG Autric Programe	<h2>Relation entre type statique et type dynamique</h2> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Le langage et ses interprétations Transfert de programmation Héritage Généralité Concurrency Interfaces graphiques Gestion des erreurs et exceptions</p> <p>Java → langage à typage statique, mais avec certaines vérifications à l'exécution</p> <ul style="list-style-type: none"> <li>À la compilation on vérifie le type des expressions 149 (analyse statique)</li> <li>Toutes les expressions sont vérifiées.</li> <li>À l'exécution, la JVM peut vérifier le type des objets 150.</li> </ul> <p>Ce type de vérification n'est pas systématique. Il a lieu seulement lors d'événements bien précis : p. ex. : lors d'un <i>downcasting</i> ou d'un test <b>instanceof</b>.</p> <p>148. C'est en fait une caractéristique habituelle des langages à typage essentiellement statique autorisant le polymorphisme par sous-type.</p> <p>149. morceaux de texte évaluables dans le programme</p> <p>150. Ces entités n'existent pas avant l'exécution, de toute façon !</p>
Compléments en POG Autric Programe	<h2>Stades de vérification et entités typables en Java</h2> <p>À la compilation : les objets</p> <p>Lors de l'instantiation d'un objet, le nom de sa classe (= son <b>type dynamique exact</b>), y est inscrit (<b>définitivement</b>). Ceci permet :             <ul style="list-style-type: none"> <li>d'exécuter des tests demandés par le programmeur (comme <b>instanceof</b>) ;</li> <li>à la méthode <b>getClass()</b> de retourner un résultat ;</li> <li>de faire fonctionner la liaison dynamique (dans <b>x.f()</b>, la JVM regarde le type de l'objet référencé par <b>x</b> avant de savoir quel <b>f()</b> exécuter) ;</li> <li>de vérifier la cohérence de certaines conversions de type :</li> </ul> </p> <pre>Object o; ... ; String s = (String)o;</pre> <p>Ceci ne concerne pas les valeurs primitives/directes : pas de place pour coder le type dans les 32 bits de la valeur directe ! (et c'est en fait inutile)</p> <p>La différence entre objets et valeurs directes provient du traitement différent du polymorphisme et des conversions de type (casts, voir plus loin).</p>
Compléments en POG Autric Programe	<h2>Notion de sous-type</h2> <p>Notion de sous-type</p> <p>Définition abstraite</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Le langage et ses interprétations Transfert de programmation Héritage Généralité Concurrency Interfaces graphiques Gestion des erreurs et exceptions</p> <p>Interprétation faible : <u>ensembliste</u>. Tout sous-ensemble d'un type donné forme un sous-type de celui-ci.</p> <p>Exemple : tout carré est un rectangle, donc le type carré est sous-type de rectangle.</p> <p>→ insuffisant car un type n'est pas un simple ensemble 156 : il est aussi muni d'opérations, d'une structure, de contrats 157, ...</p> <p>Contrat : propriété que les implémentations d'un type s'engagent à respecter. Un type honore un tel contrat si et seulement si toutes ses instances ont cette propriété.</p> <p>156. Pour les algébriques, on peut faire l'analogie avec les groupes, par exemple : un sous-ensemble d'un groupe n'est pas forcément un groupe (il faut aussi qu'il soit stable par les opérations de groupe, afin que la structure soit préservée).</p> <p>157. Formes ou informels (javadoc).</p> <p>158. Contravariance des paramètres et covariance du type de retour.</p>

Commentaires	en P00
Autor-Dégrave	Introduction
	Généralités
	Style
	Objets et classes
	Types et polymorphisme
	Structures de données
	Interactions
	Terminologie
	Résumé
	Analyses
	Conclusion
	Remerciements
	Annexe
	Index
<h2>Notion de sous-type</h2>	
<h3>Interprétations</h3>	
<p>Pourquoi le sous-type structurel est insuffisant ?</p>	
<p><b>Exemple :</b></p>	
<ul style="list-style-type: none"><li>• Dans le cours précédent, les instances de la classe <i>FiboGen</i> gènèrent la suite de Fibonacci.</li><li>• Contrat possible<sup>159</sup> : « le rapport de 2 valeurs successives tend vers <math>\varphi = \frac{1+\sqrt{5}}{2}</math> (nombre d'or) ».</li><li>• On sait prouver ce contrat pour la méthode <i>next</i> des instances directes de <i>FiboGen</i>.)</li></ul>	
<p>Orien empêche de créer un sous-type <i>BadFib</i> (sous-classe<sup>160</sup>) de <i>Fibogen</i> dont la méthode <i>next</i> retournerait toujours 0.</p>	
<p>→ Les instances de <i>BadFib</i> seraient alors des instances de <i>Fibogen</i> violant le contrat.</p>	
<hr/> <p>159. Raisonnabilé, dans le sens où c'est une propriété mathématique démontrée pour la suite de Fibonacci, qui donc doit être vraie dans toute implémentation correcte.</p>	
<p>160. Une sous-classe est bien un sous-type au sens structurel : les méthodes sont héritées.</p>	

Notion de sous-type	
Interprétations	Interprétations
Compléments en 700	Introduction
Affine dégénérante	Catégories
Implémentations	Système
Interaction	Objets et types/programe
Relations	Entités et associations
Attribut	Contraintes
Opérations	Opérations
Analyses	Analyses

- Hélas, le LSP est une notion trop forte pour les compilateurs : pour des contrats non triviaux, aucun programme ne sait vérifier une telle substituabilité (indécidable).  
**Cette notion n'est pas implémentée par les compilateurs, mais c'est bien celle que le programmeur doit avoir en tête pour écrire des programmes corrects !**
- **Interprétation en pratique** : tout langage de programmation possède un système de règles simples et vérifiables par son compilateur, définissant « **son** » sous-typage.

<h2>Notion de sous-typage</h2> <p>Interprétations : en pratique</p>	<p><b>Les grandes lignes du sous-typage selon Java :</b> (détails dans JLS 4.10 et ci-après)</p> <ul style="list-style-type: none"> <li>• Pour les 8 types primatifs, il y a une relation de sous-typage pré-définie.</li> <li>• Pour les types référence, le sous-typage est nominal : A n'est sous-type de B que si A est déclaré comme tel (<b>implements</b> ou <b>extends</b>).</li> <li>• Mais la définition d'A ne passe la compilation que si certaines contraintes structurelles<sup>162</sup> sont vérifiées, concernant les redéfinitions de méthodes.</li> <li>• Types primatifs et types référence forment deux systèmes déconnectés. Aucun type référence n'est sous-type ou supertype d'un type primitif.</li> </ul>
<a href="#">Sommaire</a> <a href="#">Index</a>	<p><small>162. Cf. cours sur les interfaces et sur l'héritage pour voir quelles sont les contraintes exactes.</small></p>

**Notion de sous-typage pour Java**

Relation de sous-typage en Java<sup>165</sup> (2)

**Types référence :**

$A <: B$  si  $B$  est **Object**<sup>166</sup> ou s'il existe des types  $A_0 (= A), A_1, \dots, A_n (= B)$  tels que pour tout  $i \in 1..n$ , une des règles suivantes s'applique :<sup>167</sup>

- (**implémentation d'interface**)  $A_{i-1}$  est une interface,  $A_i$  est une classe et  $A_{i-1}$  implémente  $A_i$ ;
- (**héritage de classe**)  $A_{i-1}$  et  $A_i$  sont des classes et  $A_{i-1}$  étend  $A_i$ ;
- (**héritage d'interface**)  $A_{i-1}$  et  $A_i$  sont des interfaces et  $A_{i-1}$  étend  $A_i$ ;
- (**covariance des types tableau**)<sup>167</sup>  $A_{i-1}$  et  $A_i$  resp. de forme  $a[]$  et  $b[]$ , avec  $a <: b$ ;

---

<sup>165</sup> C'est vrai même si  $A$  est une interface, alors même qu'aucune interface inhérite de **Object**.

<sup>166</sup> Pour être exhaustif, il manque les règles de sous-typage pour les types génériques.

<sup>167</sup> Les types tableau sont des classes (très) particulières, implémentant les interfaces **Cloneable** et **Serializable**. Donc tout type tableau est aussi sous-type de **Object**, **Cloneable** et **Serializable**.

168. Cf. JLS 1.10.

**Notion de sous-typage pour Java**

Relation pour les types références, diagramme

```

classDiagram
    class Object
    class Comparable {
        <<Comparable<<
        <<Serializable<<
    }
    class Number {
        <<Number<<
        <<Comparable<<
        <<Serializable<<
    }
    class Integer {
        <<Integer<<
        <<Number<<
        <<Comparable<<
        <<Serializable<<
    }

    Object --> Comparable
    Object --> Serializable
    Comparable <|-- Number
    Comparable <|-- Integer
    Number --> Comparable
    Number --> Serializable
    Integer --> Comparable
    Integer --> Serializable
  
```

The diagram illustrates the inheritance and interface implementation relationships in Java:

- Object** is the root class.
- Comparable** and **Serializable** are interfaces.
- Number** and **Integer** are classes that implement both **Comparable** and **Serializable**.
- Integer** is a subclass of **Number**.
- Comparable** and **Serializable** have dashed arrows pointing from **Object**, indicating they are implemented by all objects.
- Comparable** and **Serializable** also have dashed arrows pointing from **Number** and **Integer**, indicating they are implemented by these classes.
- Comparable** and **Serializable** have solid arrows pointing from **Integer**, indicating they are implemented by this class.

## Notion de sous-typage pour Java

Principe fondamental – explications

Pourquoi ce remplacement ne gène pas l'exécution :

- les objets sont utilisables sans modification comme instances de tous leurs supertypes<sup>171</sup> (**sous-typage inclusif**). P. ex : `Object o = "toto"` fonctionne.
- Java<sup>172</sup> s'autorise, si nécessaire, à remplacer une valeur primitive par la valeur la plus proche dans le type cible (**sous-typage coercitif**). P. ex : après l'affectation `float f = 1.000_000_000_123f` ; la variable `f` vaut `1.0E12` (on a perdu les derniers chiffres).

---

171. Les contraintes d'implémentation d'interface et d'héritage garantissent que les méthodes des supertypes peuvent être appelées.

172. Si nécessaire, Java convertit les constantes et insère des instructions dans le code-octet pour convertir les valeurs variables à l'exécution.

## Notion de sous-typage pour Java

### Transtypage (type casting)

Compléments en P40	Audrey Degorre
Introduction	
Généralités	
Style	
Objets et classes	
Typos et polymorphisme	
La présence d'un type	
Surcharge	
Promotion	
Surcharge	
Heritage	
Généricité	
Concurrente	
Interfaces graphiques	
Gestion des erreurs et exceptions	
Détails	

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

- Corollaires :
  - on peut affecter à toute variable une expression de son sous-type (ex : `double z = 12;`);
  - on peut appeler toute méthode avec des arguments d'un sous-type des types déclarés dans sa signature (ex : `Math.pow(3, -z);`);
  - on peut appeler toute méthode d'une classe T donnée sur un récepteur instance d'une sous-classe de T (ex : `"toto".hashCode();`).
- Ainsi, le sous-typage est la base du système de polymorphisme de Java.

173. Défiliés dans la JLS, chapitre 5.

174. On ne mentionne pas les mécanismes explicites et évidents tels que l'utilisation de méthodes portant du A et retournant du B. Si on va par là, tout type est convertible en tout autre type.

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

#### Compléments en P40

#### Audrey Degorre

### Transtypage (type casting)

#### Autres termes employés (notamment dans la JLS)

### Transtypage (type casting)

<h2>Transystype : à propos du “cast”</h2> <p>i.e., la notation “Type expr”</p> <ul style="list-style-type: none"> <li>Particulièrement utile pour le <u>downtyping</u>, mais sera à toute conversion.</li> <li>Sont <b>A</b> et <b>B</b> des types et <b>e</b> une expression de type <b>A</b>, alors l’expression “(<b>B</b>)<sup>e</sup>”</li> <ul style="list-style-type: none"> <li>est de type statique <b>B</b></li> <li>et à pour valeur (à l’exécution), autant que possible, la “même” que <b>e</b>.</li> </ul> <li>L’expression “(<b>B</b>)<sup>e</sup>” passe la compilation à condition que (au choix) :</li> <ul style="list-style-type: none"> <li><b>A</b> et <b>B</b> soient des types référence avec <b>A &lt;: B</b> ou <b>B &lt;: A</b>;</li> <li>que <b>A</b> et <b>B</b> soient des types primits tous deux différents de <b>boolean</b><sup>179</sup>;</li> <li>que <b>A</b> soit un type primitif et <b>B</b> la version emballée de <b>A</b>;</li> <li>ou que <b>B</b> soit un type primitif et <b>A</b> la version emballée d’un sous-type de <b>B</b> (combinaison implicite d’unboxing et <i>upcasting</i>).</li> </ul> <li>Même quand le programme compile, effets indésirables possibles à l’exécution : <ul style="list-style-type: none"> <li>perte d’information quand on convertit une valeur primitive.</li> <li><b>ClassCastException</b> quand on tente d’utiliser un objet en tant qu’objet d’un type qui n’a pas.</li> </ul> </li> </ul> <p><u>179.</u> NB : (<b>char</b>) (<b>byte</b>) est légal, alors qu’il n’y a pas de sous-type dans un sens ou l’autre.</p>	<h3>Subtilités du transystype</h3> <p>Cas des types primits : points d’implémentation</p> <p>Et concrètement, que font les conversions ?</p> <ul style="list-style-type: none"> <li><u>Upcasting</u> d’entier <math>\leq 32</math> bits vers <b>long</b> (<b>121</b>) : on complète la valeur en récupiant le bit de gauche 32 fois.<sup>184</sup></li> <li><u>Downtyping</u> d’entier vers entier <b>n</b> bits (<b>12b</b>, <b>12c</b>, <b>12s</b>, <b>12l</b>) : on garde les <b>n</b> bits de droite et on remplit à gauche en récupiant <math>32 - n</math> fois le bit plus à gauche restant.<sup>185</sup></li> </ul> <p><u>184.</u> Pour plus d’explications : chercher représentation des entiers en complément à 2<sup>n</sup>.  <u>185.</u> Ainsi, la valeur d’origine est interprétée modulo <math>2^n</math> sur un intervalle centré en 0.</p>	<h3>Subtilités du transystype</h3> <p>Cas des types référence</p> <ul style="list-style-type: none"> <li>Ainsi, après le cast, Java sait que l’objet « converti » est une instance du type cible<sup>187</sup>.</li> <li>Les méthodes exécutées sur un objet donné (avec ou sans <i>cast</i>), sont toujours celles de sa classe, peu importe le type statique de l’expression.<sup>188</sup></li> </ul> <p>Le cast change juste le type statique de l’expression et donc les méthodes qu’on a le droit d’appeler dessus (indépendamment de son type dynamique).</p> <p>Dans l’exemple ci-dessous, c’est bien la méthode <b>f()</b> de la classe <b>B</b> qui est appelée sur la variable <b>a</b> de type <b>A</b>:</p> <pre>class A { public void f() { System.out.println("A"); } } class B extends A { @Override public void f() { System.out.println("B"); } }  A a = new B(); // upcasting B &gt; A // ici, a: type statique A, type dynamique B a.f(); // afficher à len "B"</pre> <p><u>187.</u> Sans que l’objet n’ait jamais été modifié par le <i>cast</i> !</p> <p><u>188.</u> Principe de la <b>liaison dynamique</b>.</p>
---	--	---

Subtilités du transystpage	Cas des types primitifs : possible perte d'information (?)	Cas des types primitifs : impossible perte d'information (?)
<ul style="list-style-type: none"> <li>• Cas avec perte d'information possible :           <ul style="list-style-type: none"> <li>• tous les <i>downcastings</i> primitifs ;               <ul style="list-style-type: none"> <li>• <i>upcasting</i> de <b>int</b> vers <b>float</b><sup>180</sup>, <b>long</b> vers <b>float</b> ou <b>long</b> vers <b>double</b>;</li> <li>• <i>upcasting</i> de <b>float</b> vers <b>double</b> hors contexte <b>strictfp</b><sup>181</sup>.</li> </ul> </li> </ul> </li> <li>• Cas sans perte d'information : (= les autres cas = les "vrais" <i>upcastings</i>)           <ul style="list-style-type: none"> <li>• <i>upcasting</i> d'entier vers entier plus long;</li> <li>• <i>upcasting</i> d'entier <math>\leq</math> 24 bits vers <b>float</b> et <b>double</b>;</li> <li>• <i>upcasting</i> d'entier <math>\leq</math> 53 bits vers <b>double</b>;</li> <li>• <i>upcasting</i> de <b>float</b> vers <b>double</b> le sous contexte <b>strictfp</b>.</li> </ul> </li> </ul> <p>180. Par exemple, <b>int</b> utilise 32 bits, alors que la <b>manutient</b> de <b>float</b> n'en a que 24 (+ 8 bits pour la position de la virgule) <math>\rightarrow</math> certains <b>int</b> ne sont pas représentables en <b>float</b>.</p> <p>181. Selon implementation, mais pas de garantie. Cherchez à quoi sert ce mot-clé !</p>	<h2 data-bbox="1167 6 1197 199">Subtilités du transystpage</h2> <p data-bbox="1167 199 1197 233">Cas des types primitifs : exemples</p> <ul style="list-style-type: none"> <li>• <b>int i = 42; short s = i;</b> pour copier un <b>int</b> dans un <b>short</b>, on doit le rétroir. La valeur à convertir est incomme à la compilation <math>\rightarrow</math> ce sera fait à l'exécution. Ainsi le compilateur insère l'instruction <b>12s</b> dans le code-octet.</li> <li>• <b>short s = 42; 42 étant représentable sur 16 bits, ne demande pas de précaution particulière. Le compilateur compile "tel quel".</b></li> <li>• <b>short s = 42; int i = s;</b> comme un <b>short</b> est représenté comme un <b>int</b>, il n'y a pas de conversion à faire (<b>32t</b> n'existe pas).</li> <li>• <b>float x = 9;</b> ici on convertit une constante littérale entière en flottant. Le compilateur fait lui-même la conversion et met dans le code-octet la même chose que si on avait écrit <b>float x = 9.0f;</b></li> <li>• Mais si on écrit <b>int i = 9; float x = i;</b> c'est différent. Le compilateur ne pouvant pas convertir lui-même, il insère <b>12f</b> avant l'instruction qui va copier le sommet de la pile dans <b>x</b>.</li> </ul> <h2 data-bbox="1167 233 1197 435">Polymorphisme</h2> <p data-bbox="1167 435 1197 469">Un principe fondamental de la POO</p> <h3 data-bbox="1167 435 1197 568">Définition (Polymorphisme)</h3> <p data-bbox="1167 568 1197 649">Une instruction/une méthode/d'une classe/... est dite <b>polymorphe</b> si elle peut travailler sur des données de types concrets différents, qui se comportent de façon similaire.</p> <ul style="list-style-type: none"> <li>• Le fait pour un même morceau de programme de pouvoir fonctionner sur des types concrets différents favorise de façon évidente la <u>réutilisation</u>.</li> <li>• Or tout code réutilisé, plutôt que dupliqué quasi à l'identique, n'a besoin d'être configuré qu'une seule fois par bug détecté.</li> <li>• Donc le polymorphisme aide à la <u>bien programmer</u>, ainsi la POO en a fait un de ses "piliers".</li> </ul> <p>Il y a en fait plusieurs formes de polymorphisme en Java....</p>	

<h2>Formes de polymorphisme</h2> <p>Les 3 formes de polymorphisme en Java</p> <ul style="list-style-type: none"> <li>• polymorphisme <b>ad hoc</b> (via la surcharge) : le même code recompli dans différents contextes peut fonctionner pour des types différents.</li> </ul> <p>Attention : résolution à la compilation → après celle-ci, type concret fixé.</p> <p>Donc pas de réutilisation du code compilé → forme très faible de polymorphisme.</p> <p>→ <b>polymorphisme par sous-type</b> : le code peut être exécuté sur des données de différents sous-types d'un même type (souvent une <b>interface</b>) sans recompilation. → forme classique et privilégiée du polymorphisme en POO</p> <ul style="list-style-type: none"> <li>• <b>polymorphisme paramétré</b> :</li> </ul> <p>Concerne le code utilisant les <b>type génériques</b> (ou paramètres, cf. généralité)</p> <p>Le même code peut fonctionner, sans recompilation, quelle que soit la concrétisation des paramètres.</p> <p>Ce polymorphisme permet d'exprimer des relations fines entre les types.</p>	<p>Introduction Réalisation Style Objets et classes Types et polymorphisme Sous-type Surcharge Inheritance Généralité Documentation Interfaces Généralité Gestion de erreurs et exceptions Révision</p>
<p>Introduction Réalisation Style Objets et classes Types et polymorphisme Sous-type Surcharge Inheritance Généralité Documentation Interfaces Généralité Gestion de erreurs et exceptions Révision</p>	<p>Complément en POO Autre Doc</p> <h2>SurchARGE</h2> <p>Résolution (simplifiée...)</p> <p>Pour un appel à "<b>f</b>" donné, le compilateur :</p> <ol style="list-style-type: none"> <li>1 liste les méthodes de nom <b>f</b> du contexte courant;</li> <li>2 garde celles dont la signature subsume la signature d'appel;</li> <li>3 élimine celles dont la signature subsume la signature d'une autre candidate (ne garde que les signatures les plus spécialisées);</li> <li>4 applique quelques autres règles<sup>193</sup> pour éliminer d'autres candidates;</li> <li>5 s'il reste plusieurs candidates à ce stade, renvoie une erreur (appel ambigu);</li> <li>6 sinon, inscrit la référence de la dernière candidate restante dans le code octet.</li> </ol> <p>C'est celle-ci qui sera appelée à l'exécution.<sup>194</sup></p> <p>193. Notamment liées à l'héritage, nous ne détaillons pas.</p> <p>194. Exactement celle-ci pour les méthodes statiques. Pour les méthodes d'instance, on a juste déterminé que la méthode qui sera choisie à l'exécution aura cette signature-là. Voir liaison dynamique.</p>
<p>Introduction Réalisation Style Objets et classes Types et polymorphisme Sous-type Surcharge Inheritance Généralité Documentation Interfaces Généralité Gestion de erreurs et exceptions Révision</p>	<p>Complément en POO Autre Doc</p> <h2>SurchARGE</h2> <p>Pourquoi elle ne permet que du polymorphisme "faible"</p> <p><b>Alternative</b>, écrire la méthode, une bonne fois pour toutes, de la façon suivante :</p> <pre>public static void f(Object o) { // méthode "éeelement" polymorphe     if (o instanceof String) f((String) o);     else if (o instanceof Integer) f((Integer) o);     else /* gérer l'erreur */ }</pre> <p>Mais ici, c'est en réalité du polymorphisme par sous-type<sup>195</sup> (de Object).</p>
<p>Introduction Réalisation Style Objets et classes Types et polymorphisme Sous-type Surcharge Inheritance Généralité Documentation Interfaces Généralité Gestion de erreurs et exceptions Révision</p>	<p>Analyses</p>

<p><b>Surcharge</b> = situation où existent plusieurs définitions (au choix)</p> <ul style="list-style-type: none"> <li>• dans un contexte donné d'un programme, de plusieurs méthodes de même nom</li> <li>• dans une même classe, plusieurs constructeurs,</li> <li>• d'opérateurs arithmétiques dénotés avec le même symbole. 190 .</li> </ul> <p><b>Signature d'une méthode</b> = <math>n</math>-uplet des types de ses paramètres formels.</p> <p><b>Remarques :</b></p> <ul style="list-style-type: none"> <li>• Interdiction de définir dans une même classe 191 2 méthodes ayant même nom même signature (ou 2 constructeurs de même signature).</li> <li>• <math>\rightarrow</math> 2 entités surchargées ont forcément une signature différente 192 .</li> </ul> <p>190. P. ex. : "r" est défini pour <b>int</b> mais aussi pour <b>double</b></p> <p>191. Les méthodes héritées comptent aussi pour la surcharge. Mais en cas de signature identique, il masque et non surcharge. Donc ce qui est dit ici reste vrai.</p> <p>192. Nombre ou type des paramètres diffèrent; le type de retour ne fait pas partie de la signature et n'a rien à voir avec la surcharge !</p>	<p><b>Surcharge</b></p> <p>Adrien Degorre</p> <p>Introduction</p> <p>Généralités</p> <p>Style</p> <p>Objets et classes</p> <p>Types et polymorphisme</p> <p>La programmation orientée objet</p> <p>La programmation fonctionnelle</p> <p>Réflexion</p> <p>Surcharge</p> <p>Heritage</p> <p>Généralité</p> <p>Concurrence</p> <p>Interfaces graphiques</p> <p>Gestion des erreurs et exceptions</p> <p>Révision</p> <p>Compléments en P00</p> <p>Autre Degorre</p> <p>Introduction</p> <p>Généralités</p> <p>Style</p> <p>Objets et classes</p> <p>Types et polymorphisme</p> <p>La programmation orientée objet</p> <p>La programmation fonctionnelle</p> <p>Réflexion</p> <p>Surcharge</p> <p>Heritage</p> <p>Généralité</p> <p>Concurrence</p> <p>Interfaces graphiques</p> <p>Gestion des erreurs et exceptions</p> <p>Exemple</p>	<p><b>Surcharge</b></p> <pre>public class Surcharge {     public static void f(double x) { System.out.println("double"); }     public static void f(int x) { System.out.println("int"); }     public static void g(int x, double z) { System.out.println("int,double"); }     public static void g(double x, int z) { System.out.println("double,int"); }     public static void main(String[] args) {         f(0); // affiche "int"         f(0); // affiche "double"         g(0, 0); // ne compile pas         g(0, 0); // affiche "double int"     } }</pre>	<p><b>Interface</b></p> <p>Adrien Degorre</p> <p>Introduction</p> <p>Généralités</p> <p>Style</p> <p>Objets et classes</p> <p>Types et polymorphisme</p> <p>La programmation orientée objet</p> <p>La programmation fonctionnelle</p> <p>Réflexion</p> <p>Surcharge</p> <p>Heritage</p> <p>Généralité</p> <p>Concurrence</p> <p>Interfaces graphiques</p> <p>Gestion des erreurs et exceptions</p> <p>Exemple</p>
---	---	---	---

Interfaces : Syntaxe de la déclaration	<pre><b>public interface Comparable</b> Comparable { <b>int compareTo(Object other)</b>; }</pre> <p>Déclaration comme une classe, en remplaçant <b>class</b> par <b>interface</b>, mais :</p> <ul style="list-style-type: none"> <li>• constructeurs interdits;</li> <li>• tous les membres implicitement <sup>197</sup> <b>public</b>;</li> <li>• attributs implicitement <b>static final</b> (= constantes);</li> <li>• types membres nécessairement et implicitement <b>static</b>;</li> <li>• méthodes d'instance implicitement <b>abstract</b> (simple déclaration sans corps);</li> <li>• méthodes d'instance non-abstractes signalées par mot-clé <b>default</b>;</li> <li>• les méthodes <b>private</b> sont autorisées (annule <b>public</b> et <b>abstract</b>), autres membres obligatoirement <b>public</b>;</li> <li>• méthodes <b>final</b> interdites.</li> </ul> <p><sup>196</sup> Méthodes <b>static</b> et <b>default</b> depuis Java 8, <b>private</b> depuis Java 9.</p> <p><sup>197</sup> Ce qui est implicite n'a pas à être écrit dans le code, mais peut être écrit tout de même.</p>
Interfaces : usage (2)	<h3>Interfaces : usage (2)</h3> <p>une méthode de tri polymorphe</p> <pre>public class Tri {     static void trie(Comparable [] tab) {         /* ... algorithme de tri            utilisant tab[i].compareTo(tab[j])            ...         }          public static void main(String [] args) {             Mot [] tableau = creeTableauMotAuHasard();             // on suppose que creeTableauMotAuHasard existe             trierTableau();             // Mot [] est compatible avec Comparable []         }     } }</pre>
Interfaces : méthodes par défaut	<h3>Interfaces : méthodes par défaut</h3> <p>Exemple</p> <pre>interface ArbreBinnaire {     ArbreBinnaire gauche();     ArbreBinnaire droite();     <b>default</b> int hauteur() {         ArbreBinnaire g = gauche();         <b>int</b> hg = (g == null)? 0: g.hauteur();         ArbreBinnaire d = droite();         <b>int</b> hd = (d == null)? 0: d.hauteur();         <b>return</b> 1 + (hg&gt;hd)? hg.hauteur();     } }</pre> <p><b>Remarque</b> : on ne peut pas (re)définir par défaut des méthodes de la classe <b>Object</b> (comme <b>toString</b> et <b>equals</b>).</p> <p><b>Raison</b> : une méthode par défaut n'est là que... par défaut. Toute méthode de même nom héritée d'une classe est prioritaire. Ainsi, une implémentation par défaut de <b>toString</b> sera tout le temps ignorée.</p>
Interfaces : génération	<p>Introduction</p> <p>Généralités</p> <p>Style</p> <p>Objets et classes</p> <p>Types et polymorphisme</p> <p>La conversion en objet</p> <p>Conversion de type</p> <p>Paramétrage</p> <p>Interfaces</p> <p>Interfaces génériques</p> <p>Interfaces et énumérations</p> <p>Annotations</p> <p>Héritage</p> <p>Classe mère</p> <p>Interfaces</p> <p>Annotations</p> <p>Annotations et exceptions</p> <p>Héritage</p> <p>Génération</p> <p>Interfaces</p> <p>Classes générées</p> <p>Exceptions et erreurs</p> <p>Détails</p>

## Interfaces : syntaxe de la déclaration

Pourquoi ces contraintes ?

- Limites dues plus à l'idéologie (qui s'est assouplie) qu'à la technique. 198
- À la base : interface = juste description de la communication avec ses instances.
- Mais, dès le début, quelques « enverses » : constantes statiques, types membres.
- Java 8 permet qu'une interface contienne des implementations → la construction **interface** va au-delà du concept d'« interface » de POO.
- Java 9 ajoute l'idée que ce qui n'appartient pas à l'**interface** (selon P00) peut bien être privé (pour l'instant seulement méthodes).

**Ligne rouge pas encore franchie** : une interface ne peut pas imposer une implémentation à ses sous-types (interdis : constructeurs, attributs d'instance et méthodes **final**).

**Conséquence** : une interface n'est pas non plus directement<sup>199</sup> instanciable.

198. Il y a cependant des vraies contraintes techniques, notamment liées à l'héritage multiple.  
199. Mais très facile via classe anonyme : **new UneInterface{ ... }**.

## Interfaces : notation UML

```

classDiagram
    class Comparable {
        <<interface>>
        + compareTo(other : Object) : int
    }
    class Mot {
        - contenu : String
        + compareTo(other : Object) : int
    }
    Comparable <|-- Mot
    Comparable <|-- Mot
    
```

ou version "abîgée" :  
**Comparable**

## Compléments en POO

Autre Départie

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Concurrente  
Héritage  
Généralité  
Concurrency  
Interfaces  
gratuites  
Gestion des erreurs et exceptions  
Discussion

## Compléments en POO

Autre Départie

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Concurrente  
Héritage  
Généralité  
Concurrency  
Interfaces  
gratuites  
Gestion des erreurs et exceptions

## Compléments en POO

Autre Départie

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Concurrente  
Héritage  
Généralité  
Concurrency  
Interfaces  
gratuites  
Gestion des erreurs et exceptions

## Compléments en POO

Autre Départie

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Concurrente  
Héritage  
Généralité  
Concurrency  
Interfaces  
gratuites  
Gestion des erreurs et exceptions

**Interfaces : usage (1)**

**public interface Comparable { int compare(Object other); }**

```
class Mot implements Comparable {
    private String contenu;
    public int compareTo(Object other) {
        return ((Mot) autre).contenu.length() - contenu.length();
    }
}
```

- Mettre **implements** **I** dans l'en-tête de la classe **A** pour implémenter l'interface **I**.
- Les méthodes de **I** sont définissables dans **A**. Ne pas oublier d'écrire **public**.
- Pour obtenir une « vraie » classe (non abstraite, i.e. instanciable) : nécessaire de définir toutes les méthodes abstraites promises dans l'interface implémentée.
- Si toutes les méthodes promises ne sont pas définies dans **A**, il faut précéder la déclaration de **A** du mot-clé **abstract** (classe abstraite, non instantiable)
- Une classe peut implémenter plusieurs interfaces :

```
class A implements I, J, K { ... }.
```

**Interfaces : méthodes par défaut (Java  $\geq$  8)**

- Méthode par défaut**: méthode d'instance, non abstraite, définie dans une interface. Sa déclaration est précédée du mot-clé **default**.
- N'utilise pas les attributs de l'objet, encore inconnus, mais peut appeler les autres méthodes déclarées, même abstraites.
- Utilité : implémentation par défaut de cette méthode, héritée par les classes qui implémentent l'interface → moins de réécriture.
- Possibilité d'une forme (table) d'héritage multiple (via superclasse + interface(s) implementée(s)).
- Avertissement** : héritage possible de plusieurs définitions pour une même méthode par plusieurs chemins.  
Il sera parfois nécessaire de « désambiguier » (on en reparlera).

**Héritage d'implémentations multiples**  
A cause des méthodes par héritage des interfaces

Quand une implémentation de méthode est héritée à la fois d'une superclasse et d'une interface, c'est la version héritée de la classe qui prend le dessus.

Java n'oblige pas à éviter l'ambiguité dans ce cas.

```
interface I {
    default void f1() { System.out.println("I"); }
}

class B {
    public void f1() { System.out.println("B"); }
}

class A extends B implements I {}
```

Ce programme compile et **new A().f();** affiche **B**.

Du bon usage des interfaces

Programmez à l'interface (1)

Évitez d'écrire, dans votre programme, le nom des classes des objets qu'il utilise.

Cela veut dire, évitez :

- MaClasse**
- Madresse**

et préférez :

- Dépendance**
- Dependanceimplem**

Cela s'appelle « **programmer à l'interface** ».

<b>Du bon usage des interfaces</b> Programmmez à l'interface (2)	<p>Introduzione Generali Stile Oggetti e classi Tipi e polimorfismo Concurrente Interfacce Héritage Généralité Comportement Interface graphique Gestion des erreurs et exceptions</p> <p>Introduction Généralités Style Oggetti e classi Tipi e polimorfismo Concurrente Interfacce Héritage Généralité Comportement Interface graphique Gestion des erreurs et exceptions</p> <p>Introduction Généralités Style Oggetti e classi Tipi e polimorfismo Concurrente Interfacce Héritage Généralité Comportement Interface graphique Gestion des erreurs et exceptions</p> <p>Introduction Généralités Style Oggetti e classi Tipi e polimorfismo Concurrente Interfacce Héritage Généralité Comportement Interface graphique Gestion des erreurs et exceptions</p>
	<ul style="list-style-type: none"> <li>plutôt facile quand le nom de classe est utilisé en tant que <u>type</u> (notamment dans déclarations de variables et de méthodes)           <ul style="list-style-type: none"> <li>→ remplacer par des noms d'interfaces (ex : <code>List</code> à la place de <code>ArrayList</code>)</li> </ul> </li> <li>pour instancier ces types, il faut bien que des constructeurs soient appelés, mais :           <ul style="list-style-type: none"> <li>si vous codez une bibliothèque, laissez vos clients vous fournir vos dépendances (p. ex. : en les passant au constructeur de votre classe) → <b>injection de dépendance</b></li> </ul> </li> </ul> <pre>public class MyLib {     private final SomeInterface aDependency;     public MyLib(SomeInterface dependency) { this.aDependency = dependency; } }</pre> <ul style="list-style-type: none"> <li>sinon, circonscrire le problème en utilisant des <b>fabriques</b> 200 définies ailleurs (par vous ou par un tiers). <code>list &lt; Integer &gt; l = List.of(4, 5, 6);</code></li> </ul> <p>200. Plusieurs variantes du pattern « fabrique » envisageables, cf. GoF. Dans l'exemple : fabrique → utilisable des fabriques abstraites (<i>abstract factory</i>). Variante la plus aboutie : utiliser des fabriques abstraites.</p>

- Pourquoi programmer à l'interface :

Une classe qui mentionne par son nom une autre classe contient une dépendance statique [201] à cette dernière. Cela entraîne des rigidités.

Alors que, au contraire, une classe A programmée « à l'interface » est

- polymorphe : on peut affecter à ses attributs et passer à ses méthodes tout implémentant la bonne interface, pas seulement des instances d'une certaine classe fixée « en dur ».
  - gain en adaptabilité
- évolutif : il n'y a pas d'engagement quant à la classe concrète des objets retournés par ses méthodes.

Il est donc possible de changer leur implémentation sans « casser » les clients.

[201] – écriture « en dur », sans possibilité de s'en dégager à moins de modifier le code et de le recompiler.

util en PRO	Afin de gérer l'interaction entre les classes	Introduction Généralités Style	Généralités Style et classes	Tutoriel en développement d'applications sur smartphone	Héritage Généralité Concurrente Interfaces générales	Gestion des erreurs et exceptions
Du bon usage des interfaces	Fabriques abstraites à l'âge des lambdas	Introduction Généralités Style	Généralités Style et classes	Tutoriel en développement d'applications sur smartphone	Héritage Généralité Concurrente Interfaces générales	Gestion des erreurs et exceptions
<b>Besoin :</b> dans <code>MyClass</code> , créer des instances d'une interface <code>Dep</code> connue, mais d'implémentation inconnue à l'avance.						
<b>Réponse classique :</b> on écrit une interface <code>DepFactory</code> avec méthode <code>Dep create()</code> et on ajoute au constructeur de <code>MyClass</code> un argument <code>DepFactory factory</code> . Pour créer une instance de <code>Dep</code> on fait juste <code>factory.create()</code> .						
<b>Version moderne :</b> remplacer <code>DepFactory</code> par						

Principe d'inversion de dépendance (2) : sous forme de diagramme

```

classDiagram
    package monpackage {
        Maclasse
        Interfaçadeale
    }
    package packagelclient {
        ClasseClient
        ImplenClient
    }
    Maclasse <|-- Interfaçadeale
    ClasseClient <|-- ImplenClient
    Interfaçadeale --> Maclasse
    ClasseClient --> ImplenClient
    Interfaçadeale --> ImplenClient
  
```

Du bon usage des interfaces (2), sous forme de diagramme

(remarquer le sens des flèches entre les 2 packages)

<p>compréhension en P00</p> <p>Audrey Degorre</p> <p>Introduction Généralités Style Classes et objets Types et interfaces Sous-type Thérapie</p> <p>paramètres variables membres fonctions classes héritage Généralité Concurrence Interfaces graphiques</p> <p>Gestion des erreurs et exceptions</p>	<p>Introduction Généralités Style Classes et objets Types et interfaces Sous-type Thérapie</p> <p>paramètres variables membres fonctions classes héritage Généralité Concurrence Interfaces graphiques</p> <p>Gestion des erreurs et exceptions</p>	<p>Introduction Généralités Style Classes et objets Types et interfaces Sous-type Thérapie</p> <p>paramètres variables membres fonctions classes héritage Généralité Concurrence Interfaces graphiques</p> <p>Gestion des erreurs et exceptions</p>
<p>Du bon usage des interfaces (3)</p> <p>Principe d'inversion de dépendance (3)</p>		
<p><b>Pourquoi faire cela ?</b></p> <ul style="list-style-type: none"> <li>• l'interface écrit est idéale et facile à utiliser pour programmer la bibliothèque</li> <li>• ses évolutions restent sous le contrôle de l'auteur de la bibliothèque, qui ne peut donc plus être « cassée » du fait de quelque un d'autre</li> <li>• la bibliothèque étant « programmée à l'interface », elle sera donc polymorphe.</li> </ul>	<p><b>Pourquoi dit-on « inversion » ?</b></p> <ul style="list-style-type: none"> <li>• Parce que le code source de la bibliothèque qui dépend, à l'exécution, d'un composant supposé plus « concret »<sup>207</sup>, ne dépend pas de la classe implémentant ce dernier. Selon le DIP, c'est le contraire qui se produit (dépendance à l'interface).</li> </ul>	<p>En des termes plus savants :</p> <p>« <u>Depend upon Abstractions. Do not depend upon concretions.</u> »<sup>208</sup></p>
<p><u>207.</u> et donc d'implémentation susceptible de changer plus souvent (justification du DIP par son inventeur)</p> <p><u>208.</u> Robert C. Martin (2000) dans <i>“Design Principles and Design Patterns”</i>.</p>		

Du bon usage des interfaces

Le patron « adaptateur » (GoF) sous forme de diagramme UML

```

classDiagram
    class Client
    class ClasseClient
    class ClasseBib1 {
        <<InterfaceBib1>>
    }
    class Quasiimpl {
        <<bibliothèque2>>
    }
    class Adaptateur {
        <<+ proxy : Quasiimpl>>
    }

    Client "2" --> ClasseBib1 : 
    ClasseClient "2" --> Quasiimpl : 
    Client "2" --> ClasseClient : 
    ClasseBib1 "2" --> Client : 
    Quasiimpl "2" --> ClasseClient : 
    Client "2" --> Client
  
```



## Héritage : la racine

## Héritage : la racine

Compléments en P00  
Aidez Degorre

La classe Object

- superclasse de toutes les classes;
- superclasse directe de toutes les classes sans clause **extends** (dans ce cas, « **extends Object** » est implicite);
- racine de l'arbre d'héritage des classes <sup>223</sup>.

Et le type Object est :

- supertype de tous les types références (y compris interfaces);
- supertype direct des classes sans clause ni **extends ni implements** et des interfaces sans clause **extends**;
- unique source du graphe de sous-type des types références.

223. Ce graphe a un degré d'incidence de 1 (héritage simple) et une source unique, c'est donc un arbre. Notez que le graphe d'héritage des interfaces n'est pas un arbre mais un DAG (héritage multiple) à plusieurs sources et que le graphe de sous-type des types références est un DAG à source unique.

## Héritage : la racine

La classe Object : ses méthodes (1)

Compléments en P00  
Aidez Degorre

Object possède les méthodes suivantes :

- **boolean equals(Object other)** : teste l'égalité de **this** et **other**
- **String toString()** : retourne la représentation en **String** de l'objet <sup>224</sup>
- **int hashCode()** : retourne le « hash code » de l'objet <sup>225</sup>
- **Class<?> getClass()** : retourne l'objet-classe de l'objet.
- **protected Object clone()** : retourne un « clone » <sup>226</sup> de l'objet si celui-ci est Cloneable, sinon quitte sur exception **CloneNotSupportedException**.
- **protected void finalize()** : appelée lors de la destruction de l'objet.
- **et puis wait, notify et notifyAll** que nous verrons plus tard (cf. *Threads*).
- **Utilisé en particulier par print et pour les conversions implicites vers String dans l'opérateur < + >.**
- **Entier calculé de façon déterministe depuis les champs d'un objet, satisfaisant, par contraint,**
- **a.equals(b) ==> a.hashCode() == b.hashCode()**

224. Attention : le rapport entre **clone** et **Cloneable** est plus compliqué qu'il en a l'air, cf. EJ3 item 13.

225. Rappel du transparent précédent : si **a.equals(b)** alors il faut **a.hashCode() == b.hashCode()**.

## Héritage : ajout, redéfinition, masquage

La classe Object... (1)

Compléments en P00  
Aidez Degorre

Dans une sous-classe :

- On hérite des membres visibles <sup>228</sup> de la superclasse directe <sup>229</sup>.
- Visible = **public, protected, voire package-private**, si superclasse dans même package.
- On peut masquer (to hide) n'importe quel membre hérité :
  - méthodes : par une définition de même signature dans ce cas, le type de retour doit être identique <sup>230</sup>, sinon erreur de syntaxe!
  - autres membres : par une définition de même nom
- Les autres membres de la sous-classe sont dits ajoutés.

228. Il faut en fait visibles et non **private**. En effet : **private** est parfois visible (cf. classes imbriquées).  
229. que ceux-ci aient été directement définis, ou bien quelle les aie elle-même hérités.  
230. En fait, si le type de retour est un type référence, on peut retourner un sous-type. Par ailleurs il y a des subtilités dans le cas des types paramètres, cf. générique.

## Héritage : ajout, redéfinition, masquage

Remarques diverses (1)

Compléments en P00  
Aidez Degorre

Dans une sous-classe :

- Les membres non hérités ne peuvent pas être masqués ou redéfinis, mais rien n'empêche de définir à nouveau un membre de même nom (= ajout).

Class A { private int x; } // autorisé / Class B extends A { int x; } // avec @Override, ça ne compile pas

## Héritage : ajout, redéfinition, masquage

Le panorama... (2)

Compléments en P00  
Aidez Degorre

Un ajout simple dans un contexte peut provoquer un masquage <sup>232</sup> dans un autre :

- ```
package lib;
public class B extends A {
    public void f() { System.out.println("B"); } // avec @Override, ça ne compile pas
}
```

Les membres non hérités ne peuvent pas être masqués ou redéfinis, mais rien n'empêche de définir à nouveau un membre de même nom (= ajout).

```
Class A { private int x; } // autorisé /
Class B extends A { int x; } // avec @Override, ça ne compile pas
```

## Héritage : ajout, redéfinition, masquage

Remarques diverses (2)

Compléments en P00  
Aidez Degorre

La classe GrandParent

- La classe GrandParent protège les méthodes **visible** et **protected** de sa superclasse Parent, assure que l'héritage se fait bien.

```
class GrandParent {
    protected static int a, b; // visible et protected via GrandParent
    protected void f(); // visible et protected via GrandParent
}
```

## Héritage : ajout, redéfinition, masquage

CastorDù

Compléments en P00  
Aidez Degorre

Un ajout simple dans un contexte peut provoquer un masquage <sup>232</sup> dans un autre :

- ```
package lib;
public class B extends A {
    public void f() { System.out.println("B"); } // méthode package-private, invisible dans lib
}
```

La classe Enfant hérite **a**, **g** et **f** de Parent et **b** de GrandParent via Parent.

## Héritage : ajout, redéfinition, masquage

Remarques diverses (2)

Compléments en P00  
Aidez Degorre

Les méthodes d'instance se masquent l'une et l'autre sans se redéfinir.

- Cela a l'air de contredire ce qu'on vient de dire, mais en réalité masquer implique redéfinition seulement s'il y a masquage dans le contexte où est définie la méthode masquée.

```
class Parent {
    protected static int a, b; // visible et protected via Parent
    protected void f(); // visible et protected via Parent
}
class GrandParent {
    protected static int g(); // visible et protected via GrandParent
    protected void h(); // visible et protected via GrandParent
}

class Enfant extends Parent {
    @Override void f() { System.out.println("Enfant f"); }
    @Override void g() { System.out.println("Enfant g"); }
}
```

## Héritage : ajout, redéfinition, masquage

Exemple

Compléments en P00  
Aidez Degorre

La classe GrandParent masqués mais accessibles via préfixe GrandParent..

- a et g de GrandParent masqués mais accessibles via préfixe GrandParent..

```
class GrandParent {
    protected static int a, b; // visible et protected via GrandParent
    protected void f(); // visible et protected via GrandParent
}

class Parent {
    protected static int g(); // visible et protected via Parent
    protected void h(); // visible et protected via Parent
}

class Enfant extends Parent {
    @Override protected void f() { System.out.println("Enfant f"); }
    @Override protected void g() { System.out.println("Enfant g"); }
}
```

## Héritage : ajout, redéfinition, masquage

Remarques diverses (2)

Compléments en P00  
Aidez Degorre

La classe GrandParent masquée par celle de Parent mais peut être appellée sur un récepteur de classe GrandParent. Remarque : **super.super** n'existe pas.

- f de GrandParent héritée mais redéfinie dans Enfant. Appel de la version de Parent avec **super.f()**.

```
class GrandParent {
    protected static int a, b; // visible et protected via GrandParent
    protected void f(); // visible et protected via GrandParent
}

class Parent {
    protected static int g(); // visible et protected via Parent
    protected void h(); // visible et protected via Parent
}

class Enfant extends Parent {
    @Override void f() { System.out.println("Enfant f"); }
}
```

## Liaison statique

### Liaisons statique et dynamique

### Liaisons statique et dynamique

### Compléments en P00

Non, mais sérieusement, pourquoi distinguer la redéfinition du masque simple ?

- A la compilation :** dans tous les cas, les usages d'un nom de méthode sont traduits comme référence vers une méthode existant dans le contexte d'appel (classe).
- A l'exécution :**
  - Autres membres que méthodes d'instance : la méthode trouvée à la compilation sera effectivement appellée.
  - Mécanisme de **liaison statique** (ou précoce).
- Méthodes d'instance<sup>233</sup> : une méthode redéfinissant la méthode trouvée à la compilation sera recherchée depuis le contexte de la classe de l'objet récepteur.
- Mécanisme de **liaison dynamique** (ou tardive).

Le résultat de cette recherche peut être différent à chaque exécution.  
Ce mécanisme permet au polymorphisme par sous-type de fonctionner.

233. Sauf méthodes privées et sauf appel avec préfixe "super". → liaison statique.

### Compléments en P00

Attribut Dégorge

## Liaison dynamique

## Méthodes et classes finales

Méthodes et classes finales

## Liaison dynamique

## Méthodes et classes finales

On peut déclarer une méthode avec modificateur **final**<sup>244</sup>. Exemple :

```
class Employe {
    private String name;
    public final String getName() { return name; }
    ...
}
```

Attention aux "redéfinitions ratées" : ça peut compiler mais...

- si on se trompe dans le type ou le nombre de paramètres, ce n'est pas une redéfinition<sup>245</sup>, mais un ajout de méthode surchargée. Erreur typique :
- ```
public class Object { void f // la "vraie", c.-à-d. java.lang.Object
    ...
}
public boolean equals(Object obj) { return this == obj; }
```

Recommandé : placer l'annotation @Override devant une définition de méthode pour demander au compilateur de générer une erreur si ce n'est pas une redéfinition.

Exemple : @Override public boolean equals(Object obj){ ... }

même pas un masque

## Méthodes et classes abstraites

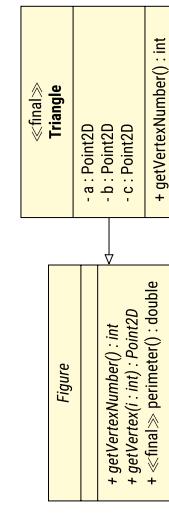
Méthodes et classes finales ou abstraites

Notation UML

## Méthodes et classes abstraites

## Méthodes et classes finales ou abstraites

Notation UML



## abstract et final

le bon usage pour les classes (1)

Constat : une classe non finale correspond à une implémentation complétée.

Idéologie : si c'est complétable c'est que c'est donc probablement incomplet.<sup>246</sup>

Si cela est vrai, alors une classe ni finale ni abstraite est louche!<sup>249</sup>. Comme **abstract + final** est exclus d'office, toute classe devrait alors être soit (juste) **abstract** soit (juste) **final**.

| Compléments en P00                                                                                      | Attrib. Dégrée                    | abstract     | final    |
|---------------------------------------------------------------------------------------------------------|-----------------------------------|--------------|----------|
| • Pourquoi contraindre ? → pour empêcher une utilisation incorrecte non prévue (cf. exemples ci-après). | le bon usage pour les classes (1) | pas abstract | abstract |

Plus précisément :

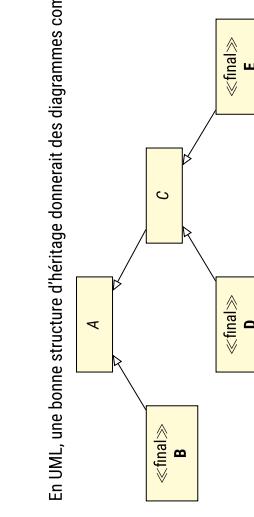
- **final**, en figurant les méthodes (une ou toutes) d'une classe, permet d'assurer des propriétés qui resteront vraies pour toutes les instances de la classe.

• **abstract** (appliquée à une classe<sup>247</sup>) empêche qu'une classe qui ne représenterait qu'une implémentation incomplète ne soit instantiée directement.

## abstract et final

le bon usage pour les classes : illustration en UML

En UML, une bonne structure d'héritage donnerait des diagrammes comme celui-ci :



## abstract et final

le bon usage pour les classes : illustration en UML

En UML, une bonne structure d'héritage donnerait des diagrammes comme celui-ci :



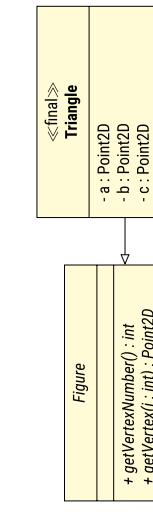
248. Ce n'est pas toujours vrai : certaines classes proposent un comportement par défaut tout à fait valable, tout en laissant la porte ouverte à des modifications (cf. composants Swing).

249. On parle de code smell. Cela dit, c'est « louche », mais pas absurde, cf. remarque précédente.

## Méthodes et classes abstraites

## Méthodes et classes abstraites

Notation UML



## abstract et final

le bon usage pour les classes (2)

Constat : une classe non finale correspond à une implémentation complétée.

Idéologie : si c'est complétable c'est que c'est donc probablement incomplet.<sup>248</sup>

Si cela est vrai, alors une classe ni finale ni abstraite est louche!<sup>249</sup>. Comme **abstract + final** est exclus d'office, toute classe devrait alors être soit (juste) **abstract** soit (juste) **final**.

| Compléments en P00                                                                                      | Attrib. Dégrée                    | abstract  | final  |
|---------------------------------------------------------------------------------------------------------|-----------------------------------|-----------|--------|
| • Pourquoi contraindre ? → pour empêcher une utilisation incorrecte non prévue (cf. exemples ci-après). | le bon usage pour les classes (2) | pas final | louche |

Plus précisément :

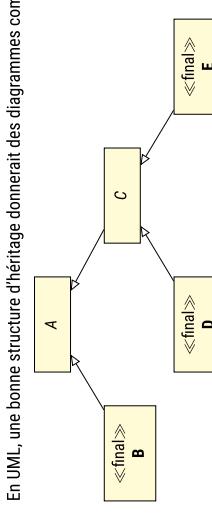
- **final**, en figurant les méthodes (une ou toutes) d'une classe, permet d'assurer des propriétés qui resteront vraies pour toutes les instances de la classe.

• **abstract** (appliquée à une classe<sup>247</sup>) empêche qu'une classe qui ne représenterait qu'une implémentation incomplète ne soit instantiée directement.

## abstract et final

le bon usage pour les classes : illustration en UML

En UML, une bonne structure d'héritage donnerait des diagrammes comme celui-ci :



## abstract et final

le bon usage pour les classes : illustration en UML

En UML, une bonne structure d'héritage donnerait des diagrammes comme celui-ci :



248. Ce n'est pas toujours vrai : certaines classes proposent un comportement par défaut tout à fait valable, tout en laissant la porte ouverte à des modifications (cf. composants Swing).

249. On parle de code smell. Cela dit, c'est « louche », mais pas absurde, cf. remarque précédente.

## abstract et final

le bon usage pour les méthodes

## abstract et final

le bon usage pour les méthodes : exemple

## abstract et final

le bon usage pour les classes : exemple

## abstract et final

le bon usage pour les méthodes : contre-exemple

### Exemple (à ne pas faire !)

```
class Personne {
    public String getNom() { return null; } // mauvaise implémentation par défaut
}

class PersonnalImpl extends Personne {
    private String nom;
    @Override public String getNom() { return nom; }
}
```

### Mieux :

```
abstract class Personne {
    public abstract String getNom();
}

final class PersonneImpl extends Personne {
    private String nom;
    @Override public String getNom() { return nom; }
}
```

Récursion non bornée ! → StackOverflowError.

### À ne pas faire non plus :

```
class Personne {
    private String nom; // return person.nom; // il faudrait final
    public String getNom() { return nom; } // à aussi
    public String getNomCancel() { return null; } // + getNom() ; // appel à méthodes redéfinissables
    return getNom(); // appels mutuellement récursifs non voulus.
}
```

### Sans final, Personne est une classe de base fragile. Quelqu'un pourrait écrire :

```
class Personne2 extends Personne {
    @Override public String getNom() { return getNomComple() ; split("...")[0] }
}

... puis exécuter new Personne2(...).getNom(), qui appelle getNomComple(), qui appelle getPrenom() et getNom(), qui appellent getNomComple() qui appelle...
```

250. Cela vaut aussi pour les appels de méthodes depuis une méthode **default** dans une interface.

## Compléments en PdO

Aidez Degorre

## abstract et final

le bon usage pour les classes : exemple

## abstract et final

le bon usage pour les méthodes : exemple

## abstract et final

le bon usage pour les méthodes : contre-exemple

### Quand on programme une classe extensible :

- Si possible, éviter tout appel, depuis une autre méthode de la classe 250, de méthode redéfinissable (= non **final** « ouverte »).
- À défaut le signaler dans la documentation.
- Objectif : éviter des erreurs bêtes dans les futures extensions.
- Par exemple : appels mutuellement récursifs non voulus.
- La documentation devra donner une spécification des méthodes redéfinissables assurant de conserver un comportement globalement correct.

250. Cela vaut aussi pour les appels de méthodes depuis une méthode **default** dans une interface.

## Compléments en PdO

Aidez Degorre

## abstract et final

le bon usage : résumé

## abstract et final

l'héritage casset-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

## Compléments en PdO

Aidez Degorre

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

l'héritage casse-t-il l'encapsulation ?

## abstract et final

les types scellés

##

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h2>Types finis</h2> <p>Pour quoi faire ?</p> <p>Un <b>type fini</b> est un type ayant un ensemble fini d'instances, toutes définies statiquement dès l'écriture du type, sans possibilité de créer de nouvelles lors de l'exécution.<sup>256</sup></p> <p>Certaines variables ont, en effet, une valeur qui doit éster dans un ensemble fini, prédéfini :</p> <ul style="list-style-type: none"> <li>• les 7 jours de la semaine</li> <li>• les 4 points cardinaux</li> <li>• les 3 (ou 4 ou plus) états de la matière</li> <li>• les n états d'un automate fini (dans protocole ou processus industriel, par exemple)</li> <li>• les 3 mousquetaires, les 7 nains, les 9 nazgûl...</li> </ul> <p>→ Situation intéressante car, théoriquement, <u>nombre fini de cas à tester/à vérifier.</u></p> <hr/> <p><sup>256</sup> C'est donc un type scellé (ties contraint) : clairement, si un type n'est pas scellé, il ne peut pas être fini.</p> | <h2>Classes d'énumération</h2> <pre>public enum ETAT { SOLIDE, LIQUIDE, GAZ, PLASMA }</pre> <p>Une <b>classe d'énumération</b> (ou juste énumération) est une classe particulière déclarée par un bloc syntaxique <b>enum</b>, dans lequel est donnée la liste (exhaustive et définitive) des instances (= « constantes » de l'<b>enum</b>).<br/>Elle définit un <b>type énuméré</b>, qui est un type :</p> <ul style="list-style-type: none"> <li>• <u>fini</u> : c'est la raison d'être de cette construction;</li> <li>• pour lequel l'opérateur « <code>==</code> » teste bien l'égalité sémantique<sup>260</sup> (toutes les instances représentent des valeurs différentes);</li> <li>• utilisable en argument d'un bloc <b>switch</b>;</li> <li>• et dont l'ensemble des instances s'ère facilement :</li> </ul> <pre>for (MonEnum val: MonEnum.values()) {...}</pre> <hr/> <p><sup>260</sup>. Pour les enums, identité et égalité sont synonymes.</p> | <p>Compléments en P00</p> <p>Aldrin Degorre</p> <p>Introduction</p> <p>Généralités</p> <p>Style</p> <p>Objets et classes</p> <p>Type et polymorphisme</p> <p>Héritage</p> <p>interfaces et méthodes</p> <p>énumérations</p> <p>classes et méthodes</p> <p>composants</p> <p>concurrente</p> <p>interfaces</p> <p>gestion des erreurs</p> |
| <p>Compléments en P00</p> <p>Aldrin Degorre</p> <p>Introduction</p> <p>Généralités</p> <p>Style</p> <p>Objets et classes</p> <p>Type et polymorphisme</p> <p>Héritage</p> <p>interfaces et méthodes</p> <p>énumérations</p> <p>classes et méthodes</p> <p>composants</p> <p>concurrente</p> <p>interfaces</p> <p>gestion des erreurs</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | <h2>Énumérations</h2> <p>Extensions</p> <hr/> <p>On peut donc y ajouter des membres, en particulier des méthodes :</p> <pre>public enum Day {     SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;     public boolean isWorkday() {         switch (this) {             case SUNDAY:             case SATURDAY:                 return false;             default:                 return true;         }     }     public static void main(String[] args) {         for (Day d : Day.values()) {             System.out.println(d + ":" + (d.isWorkday() ? "work" : "sleep"));         }     } }</pre>                                                                                                                                                                                                                                                                                                                                        | <p>Compléments en P00</p> <p>Aldrin Degorre</p> <p>Introduction</p> <p>Généralités</p> <p>Style</p> <p>Objets et classes</p> <p>Type et polymorphisme</p> <p>Héritage</p> <p>interfaces et méthodes</p> <p>énumérations</p> <p>classes et méthodes</p> <p>composants</p> <p>concurrente</p> <p>interfaces</p> <p>gestion des erreurs</p> |

| Compléments en 100     |                        |
|------------------------|------------------------|
| Introduction           | Alors, Degré           |
| Generalités            | Style                  |
| Définitions et classes | Objets et classes      |
| Type et polymorphisme  | Héritage               |
| Interfaces             | Énumérations           |
| Gestion des exceptions | Finalisation           |
| Concurrente            | Concurrency            |
| Interfaçage            | Interfaces             |
| Gestion des ressources | Gestion des ressources |
| Compléments en 100     | Compléments en 100     |
| Introduction           | Alors, Degré           |
| Generalités            | Style                  |
| Définitions et classes | Objets et classes      |
| Type et polymorphisme  | Héritage               |
| Interfaces             | Énumérations           |
| Gestion des exceptions | Finalisation           |
| Concurrente            | Concurrency            |
| Interfaçage            | Interfaces             |
| Gestion des ressources | Gestion des ressources |

**Écrire un type fini**

Comment faire ?

- **Mauvaise idée :** réserver un nombre fini de constantes dans un type existant (ça ne résout pas les problèmes évoqués précédemment).
- **Remarque :** c'est ce que fait la construction **enum** du langage C. Les constantes déclarées sont en effet des **int**, et le type créé est un alias de int.
- On a déjà vu qu'il fallait créer un nouveau type.
- Il faut qu'il soit impossible d'en créer des instances en dehors de sa déclaration...
  - ... qu'elles soient directes (appel de son constructeur) ou indirectes (*via* extension).
- **Bonne idée :** implémenter le type fini comme classe à constructeurs privés et créer les instances du type fini comme constantes statiques de la classe :

```
public class Piece { / peut être final... mais le constructeur privé suffit
    private Piece() {}
    public static final Piece FILE = new Piece();
    public static final Piece FACE = new Piece();
}
```

→ les **enum** de Java sont du sacré syntaxique pour écrire cela (+ méthodes utiles).

## Énumération = classe particulière

Il est possible d'écrire une classe équivalente sans utiliser le mot-clé **enum**<sup>261</sup>.

L'exemple précédent pourrait (presque<sup>262</sup>) s'écrire :

```
public final class Day extends Enum<Day> {
    public static final Day SUNDAY = new Day("SUNDAY", 0);
    public static final Day MONDAY = new Day("MONDAY", 1);
    public static final Day TUESDAY = new Day("TUESDAY", 2);
    public static final Day WEDNESDAY = new Day("WEDNESDAY", 3);
    public static final Day THURSDAY = new Day("THURSDAY", 4);
    public static final Day FRIDAY = new Day("FRIDAY", 5);
    public static final Day SATURDAY = new Day("SATURDAY", 6);
    private Day(String name, int ordinal) {
        super(name, ordinal);
    }
    // plus méthodes statiques values() et values()
}
```

**Enum<E>** est la superclasse directe de toutes les classes déclarées avec un bloc **enum**. Elle contient les fonctionnalités communes à toutes les énumérations.

261. Puisque c'est du sacré syntaxique!

262. En réalité ceci ne compile pas : javac n'autorise pas le programmeur à étendre la classe **Enum** à la main. Cela est réservé aux vraies **enum**. Si on voulait vraiment toutes les fonctionnalités des **enum**, il faudrait réécrire les méthodes de la classe **Enum**.

## Énumérations

Extensions

Chaque déclaration de constante énumérée peut être suivie d'un corps de classe, afin d'ajouter des membres ou de redéfinir des méthodes juste pour cette constante.

```
public enum Day {
    SUNDAY { @Override public boolean isWorkDay() { return false; } },
    MONDAY { @Override public boolean isWorkDay() { return false; } },
    TUESDAY { @Override public boolean isWorkDay() { return false; } },
    WEDNESDAY { @Override public boolean isWorkDay() { return false; } },
    THURSDAY { @Override public boolean isWorkDay() { return false; } },
    FRIDAY { @Override public boolean isWorkDay() { return true; } },
    SATURDAY { @Override public boolean isWorkDay() { return false; } },
    public static void main(String[] args) {
        for (Day d : Day.values()) {
            System.out.println(d + ":" + (d.isWorkDay() ? "work" : "sleep"));
        }
    }
}
```

Dans ce cas, la constante est l'instance unique d'une sous-classe<sup>263</sup> de l'**enum**.

**Remarque :** comme d'habitude, toute construction basée sur des **@Override** et la liaison dynamique est à préférer à un **switch** (quand c'est possible et que ça a du sens).

|                                                                                      |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|--------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h2>Énumérations</h2> <p>Quand les utiliser</p>                                      | <h3>Énumérations</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <ul style="list-style-type: none"> <li>Tous les types énumérés étendent la classe <code>Enum</code><sup>264</sup>.</li> <li>Donc une énumération ne peut étendre aucune autre classe.</li> <li>En revanche rien n'interdit d'écrire <code>enum Truc implements Machin { ... }</code>.</li> <li>Les types enum sont des classes à constructeur(s) privé(s).<sup>265</sup></li> <li>Ainsi aucune instance autre que les constantes déclarées dans le bloc <code>enum</code> pourra jamais exister.<sup>266</sup></li> <li>On ne peut donc pas non plus étendre un type énuméré.<sup>267</sup></li> </ul> <p>264. Version exacte : l'énumération <code>E</code> étend <code>Enum&lt;E&gt;</code>. Voir la générique.</p> <p>265. Elles sont même des <code>final</code>, si aucune des constantes énumérées n'est munie d'un corps de classe.</p> <p>266. Ainsi, toutes les instances d'une <code>enum</code> sont connues dès la compilation.</p> <p>267. On ne peut pas l'étendre « à la main », mais des sous-classes (singletons) sont compilées pour les constantes de l'enum qui sont munies d'un corps de classe.</p> |
| <h2>Énumérations et sous-type</h2> <p>Types énumérées et sous-type</p>               | <h3>Énumérations</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <ul style="list-style-type: none"> <li>Les <code>enums</code> sont bien pensés et robustes. Il est assez difficile de mal les utiliser.</li> <li>Piège possible : compter sur les ordinaux (<code>int</code> retourné par <code>ordinal()</code>) ou l'ordre relatif des constantes d'une <code>enum</code> → fragilité en cas de mise à jour de la dépendance fourniant l'<code>enum</code>.</li> </ul> <p><b>Bonne pratique pour utiliser une enum fournie par un tiers :</b> (EJ3 item 35) ne compter ni sur le fait qu'une constante possède un ordinal donné, ni sur l'ordre relatif des ordinaux (= ordre des constantes dans tableau <code>values()</code>).</p> <p>268. Vous savez, ces entiers qu'on manipule bit à bit via les opérateurs <code>&lt;&lt;</code>, <code>&gt;&gt;</code>, <code> </code>, <code>&amp;</code> et <code>~</code> et dont les programmeurs en C sont si friands...</p>                                                                                                                                                                                                               |
| <h2>Énumérations</h2> <p>Bien les utiliser</p>                                       | <h3>Énumérations</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <ul style="list-style-type: none"> <li>Les <code>enums</code> sont bien pensés et robustes. Il est assez difficile de mal les utiliser.</li> <li>Piège possible : compter sur les ordinaux (<code>int</code> retourné par <code>ordinal()</code>) ou l'ordre relatif des constantes d'une <code>enum</code> → fragilité en cas de mise à jour de la dépendance fourniant l'<code>enum</code>.</li> </ul> <p><b>Bonne pratique pour utiliser une enum fournie par un tiers :</b> (EJ3 item 35) ne compter ni sur le fait qu'une constante possède un ordinal donné, ni sur l'ordre relatif des ordinaux (= ordre des constantes dans tableau <code>values()</code>).</p> <p>268. Vous savez, ces entiers qu'on manipule bit à bit via les opérateurs <code>&lt;&lt;</code>, <code>&gt;&gt;</code>, <code> </code>, <code>&amp;</code> et <code>~</code> et dont les programmeurs en C sont si friands...</p>                                                                                                                                                                                                               |
| <h2>Autour de l'héritage</h2> <p>Le bon usage de l'héritage de classe (1)</p>        | <h3>Autour de l'héritage</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p>On a coutume de dire que :</p> <ul style="list-style-type: none"> <li>la classe <code>B</code> hérite de la classe <code>A</code> seulement si un <code>B</code> est un <code>A</code>.</li> <li>on compose<sup>269</sup> <code>A</code> dans <code>B</code> quand un <code>B</code> possède un <code>A</code>.</li> </ul> <p>Mais attention, « est un » peut être interprété de plusieurs façons :</p> <ul style="list-style-type: none"> <li>une instance de <code>A</code> peut être utilisée à la place d'<code>B</code> d'une instance de <code>B</code> ↔ <u>sous-type</u>.</li> <li>une instance de <code>A</code> est faite comme une instance de <code>B</code> ↔ <u>héritage</u>.</li> </ul> <p>269. Il s'agit juste d'utiliser une instance de <code>A</code> comme attribut de <code>B</code>. On repart de composition juste après.</p> <p>270. Même champs en mémoire, appel du constructeur parent, même code sauf si redéfini.</p>                                                                                                                                                             |
| <h2>Énumérations et collections (1)</h2> <p>Compléments en PdO<br/>Aidez Degorre</p> | <h3>Énumérations et collections (1)</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p>Il existe des implémentations de collections optimisées pour les énumérations.</p> <ul style="list-style-type: none"> <li><code>EnumSet&lt;E extends Enum&lt;E&gt;</code>, qui implémente <code>Set&lt;E&gt;</code> : on représente un ensemble de valeurs de l'énumération <code>E</code> par un champ de bits (le bit n° i vaut 0 si la constante d'ordinal i est dans l'ensemble, 1 sinon). Cette représentation est très concise et très rapide.</li> </ul> <p>Création via méthodes statiques</p> <pre>Set&lt;DAY&gt; weekend = EnumSet.of(Day.SATURDAY, Day.SUNDAY), voire Set&lt;DAY&gt; week = EnumSet.allOf(Day.<b>values</b>());</pre> <p>L'usage d'<code>EnumSet</code> est à préférer à l'usage direct des champs de bits (EJ3 item 36). On gagne en clarté et en sécurité.</p> <p>268. Vous savez, ces entiers qu'on manipule bit à bit via les opérateurs <code>&lt;&lt;</code>, <code>&gt;&gt;</code>, <code> </code>, <code>&amp;</code> et <code>~</code> et dont les programmeurs en C sont si friands...</p>                                                                     |
| <h2>Autour de l'héritage</h2> <p>Le bon usage de l'héritage de classe (2)</p>        | <h3>Autour de l'héritage</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p>Ainsi, il est envisageable d'étendre une classe <code>A</code> existante par une classe <code>B</code> seulement lorsque :</p> <ul style="list-style-type: none"> <li><code>B</code> doit pouvoir être utilisée à la place de <code>A</code>,</li> <li>l'implémentation de <code>B</code> semble pouvoir se baser sur celle de <code>A</code>,</li> <li>et <code>A</code> est fait telle que l'héritage est possible.</li> </ul> <p>C.-à-d. : si <code>A</code> ni <code>B</code> les méthodes à redéfinir ne sont <code>final</code> et le code à ajouter ou modifier est en accord avec les instructions données dans la documentation de <code>A</code>.<sup>271</sup></p> <p>Mais...</p> <p>271. Faute de documentation, évitez les constructions fragiles, comme par exemple, appeler une méthode héritée <code>f</code> depuis une méthode <code>g</code> de <code>B</code>. En effet : sauf indication contraire, <code>f</code> est susceptible d'appeler <code>g</code> → risque de <code>StackOverflowError</code>.</p>                                                                              |
| <h2>Autour de l'héritage</h2> <p>Le bon usage de l'héritage de classe (2)</p>        | <h3>Autour de l'héritage <p>Compléments en PdO<br/>Aidez Degorre</p> <p>... ce n'est pas parce qu'on peut le faire que C'est une bonne idée!</p> <ul style="list-style-type: none"> <li>Si la classe <code>B</code> hérite de <code>A</code>, elle récupère toutes les fonctionnalités héritées de <code>A</code>, y compris celles qui n'auraient pas de rapport avec l'objectif de <code>B</code>.<sup>272</sup> (Très utile pour le sous-type, mais la vraie question est : est-ce mon intention de créer un sous-type ? Cette classe sera-t-elle utilisée dans un contexte polymorphe ?)</li> <li>Les instances de <code>B</code> contiennent tous les champs de <code>A</code> (y compris privés), même devenus inutiles → « surjoués » et risque d'incohérences.</li> <li>Étendre une classe qui n'était pas conçue pour cela expose à des comportements inattendus<sup>273</sup> (non documentés par son auteur... qui n'avait pas prévu ça!).</li> </ul> <p>272. Et c'est le cas maintenant, ce ne sera peut-être pas vrai dans la prochaine version de <code>A</code>.</p> <p>273. Cf. cas de la « classe de base fragile » vu précédemment.</p> </h3>                             |

**Autour de l'héritage**

Alternatives à l'héritage de classe

Pour des objectifs simples préférer des techniques alternatives :

- créer du sous-type → implémentation d'interface <sup>274, 275</sup>
- Une interface ne crain pas le syndrome de la « classe de base fragile ».
- réutiliser des fonctionnalités déjà programmées → composition <sup>277</sup> (utiliser un objet auxiliaire possédant les fonctionnalités voulues pour les glouter à votre classe).

Ici aussi, on ne risque pas de « perturber » des fonctionnalités déjà programmées.

```

classDiagram
    class VotreClasse {
        - helper : ClasseExistante
        + méthodes utilisant helper
    }
    ClasseExistante
    VotreClasse <|-- ClasseExistante
  
```

274. Obligation et assumé dans les langages comme Rust, où l'héritage ne crée pas de sous-type.

275. EJS 20 : « Prefer interfaces to abstract classes »

276. Faux en cas de méthodes **default** → même besoin de documentation que pour l'héritage de classe.

277. F13.18 : « Favor composition over inheritance »

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Composition</b> : utilisation d'un objet à l'intérieur d'un autre pour <u>réutiliser</u> les fonctionnalités codées dans le premier. Exemple :</p> <hr/> <pre> <b>class</b> Vendeur {     <b>private double</b> marge;     <b>public Vendeur(double marge)</b> { <b>this</b>.marge = marge; }     <b>public double vend(Bien b)</b> { <b>return</b> (<b>1.</b> + marge) * <b>b.getPrixRevente()</b>; } }  <b>class</b> Boutique {     <b>private Vendeur vendeur;</b>     <b>private final List&lt;Bien&gt; stock = new ArrayList&lt;&gt;();</b>     <b>private double caisse = 0;</b>     <b>public Boutique(Vendeur vendeur)</b> { <b>this</b>.vendeur = vendeur; }      <b>public void vend(Bien b)</b> {         <b>if</b> (<b>stock.contains(b)</b>) { <b>stock.remove(b); caise += vendeur.vend(b);</b> }     } } </pre> <hr/> <p>Boutique réutilise des fonctionnalités de Vendeur sans en être un sous-type. Ces mêmes fonctionnalités pourraient aussi être réutilisées par une autre classe SuperMarche.</p> | <p>Autour de l'héritage</p> <p>Alternative : ce qu'on entend par « composition »</p> <hr/> <p><b>Composition</b> : utilisation d'un objet à l'intérieur d'un autre pour <u>réutiliser</u> les fonctionnalités codées dans le premier. Exemple :</p> <hr/> <pre> <b>class</b> Vendeur {     <b>private double</b> marge;     <b>public Vendeur(double marge)</b> { <b>this</b>.marge = marge; }     <b>public double vend(Bien b)</b> { <b>return</b> (<b>1.</b> + marge) * <b>b.getPrixRevente()</b>; } }  <b>class</b> Boutique {     <b>private Vendeur vendeur;</b>     <b>private final List&lt;Bien&gt; stock = new ArrayList&lt;&gt;();</b>     <b>private double caisse = 0;</b>     <b>public Boutique(Vendeur vendeur)</b> { <b>this</b>.vendeur = vendeur; }      <b>public void vend(Bien b)</b> {         <b>if</b> (<b>stock.contains(b)</b>) { <b>stock.remove(b); caise += vendeur.vend(b);</b> }     } } </pre> <hr/> <p>Boutique réutilise des fonctionnalités de Vendeur sans en être un sous-type. Ces mêmes fonctionnalités pourraient aussi être réutilisées par une autre classe SuperMarche.</p> |
| <p>Complémentaire<br/>en POO</p> <p>Autre Diaporama</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      | <p>Introduction</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <p>Généralités</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | <p>Style</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <p>Types et polymorphisme</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | <p>Héritage</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <p>Interaction entre classes</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             | <p>Interaction entre méthodes</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| <p>Utilisation et gestion des exceptions</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | <p>Utilisation et gestion des exceptions</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <p>Finalisation et destruction des objets</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | <p>Finalisation et destruction des objets</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <p>Débuter</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               | <p>Établir la liste</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <p>Concurrente</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | <p>Interfaces graphiques</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| <p>Interfacing</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | <p>Gestion des événements</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                                                                            |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| <p><b>Objectif :</b> ne pas hériter d'un « bagage » inutile</p> <p>Exemple problématique</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | <p>Mathématiquement les entiers sont sous-type des rationnels. Mais comment le coder ?</p> |
| <p><b>Pas terrible :</b></p> <pre data-bbox="171 89 262 586"> public class Rationnel {     private final int numerateur, denominateur;     public Rationnel(int p, int q) { numerateur = p; denominateur = q; }     // + getters et opérations } public class Entier extends Rationnel { public Entier(int n) { super(n, 1); } }</pre> <p>Ici, toute instance d'entiers contient 2 champs (certes non visibles) : numérateur et dénominateur. Or 1 seul <b>int</b> aurait dû suffire.</p> <ul style="list-style-type: none"> <li>→ utilisation trop importante de mémoire (pas très grave)</li> <li>→ risque d'incohérence à cause de la redondance (plus grave)</li> </ul> <p><b>Autre problème :</b> la classe <b>Rationnel</b> visant à être immuable (attributs <b>final</b>) serait typiquement <b>final</b> (pour empêcher des sous classes avec attributs modifiables).</p> |                                                                                            |

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |                                                                                                                                                                                |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p><b>Mémo :</b></p> <hr/> <pre> <b>public interface</b> Rationnel { <b>int</b> getNume(); <b>int</b> getDenom(); // + opérations }  <b>public interface</b> Entier extends Rationnel {     <b>int</b> getVal();     <b>default</b> <b>int</b> getNum() { <b>return</b> getVal(); }     <b>default</b> <b>int</b> getDenom() { <b>return</b> 1; }  <b>public final class</b> RationnelImmutable <b>implements</b> Rationnel {     <b>private final</b> <b>int</b> numerateur, denominateur;     <b>public</b> RationnelImmutable(<b>int</b> p, <b>int</b> q) { numerateur = p; denominateur = q; }      // + getteurs et opérations }  <b>public final class</b> EntierImmutable <b>implements</b> Entier {     <b>private final</b> <b>int</b> intValue;     <b>public</b> EntierImmutable (<b>int</b> n) { intValue = n; }      @Override <b>public</b> <b>int</b> getValuer() { <b>return</b> intValue; } } </pre> <hr/> | <p>Ainsi nos types existent en version immuable (via les classes) et en version à mutabilité non précisée (via les interfaces) le tout sans trainer de « bagage » inutile.</p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Solutions (1)**

**Objetif** : ne pas laisser l'héritage casser l'encapsulation

**Point faible** : ne réalise toujours pas à certaines extensions

---

```
class ReelPosition {
    double valeur; / et hop, on remplace l'attribut de la superclasse
    public void setValeur(arondi) { this.valeur = arondi; }
    public void getValeur() { return valeur; }
    public void setValeur(double valeur) { this.valeur = Math.floor(valeur); }
}
```

---

Ici, **racine** peut toujours retourner NaN. Pourquoi ?

2 La solution la plus précise : rendre **valeur** privé et passer **getValeur** en **final**. Cette solution garantit que le contrat sera respecté par toute classe dérivée.

**Point fort** : on restreint le strict nécessaire pour assurer le contrat.

**Point faible** : il faut réfléchir, sinon on a vite fait de manquer une faille.

| Objectif : ne pas laisser l'héritage casser l'encapsulation                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           | Solutions (2)                                                                                                                                                                                                                        |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>❸ La sûre, simple mais rigide : valeur → <b>private</b>, ReelPositif → <b>final</b>.</p> <p><b>Point fort :</b> sans faille et très facile</p> <p><b>Point faible :</b> on ne peut pas créer de sous-classe <b>ReelPositifArrondi</b>, mais on peut contourner grâce à la composition (on perd le sous-type) :</p> <pre>class ReelPositifArrondi {     private ReelPositif valeur;     public ReelPositif final valeur() { this.valeur = new ReelPositif(Math.floor(valeur)); }     public void getValeur() { return valeur; }     public void setValeur(double valeur) {         this.valeur = new ReelPositif(Math.floor(valeur));     } }</pre> | <p>Pour retrouver le polymorphisme : écrire une interface commune à implémenter (argument supplémentaire pour toujours programmer à l'interface), → on a alors mis en œuvre le patron de conception « <u>Décorateur</u> » (GoF).</p> |

**Objectif :** ne pas laisser l'héritage casser l'encapsulation

**Le patron décorateur [2]**

**Principe du patron décorateur :** on implémente un type en utilisant/composant un objet qui est déjà instance de ce type, mais en lui ajoutant de nouvelles responsabilités.

**L'intérêt :** on peut décorer plusieurs fois un même objet avec des décorateurs différents.

```

classDiagram
    class Reel {
        valeur : Nombre
    }
    class Positif {
        +getValeur()
        +setValeur(val : double)
    }
    class Nombre {
        +getValeur()
        +setValeur(val : double)
    }
    interface Nombre {
        <<interface >>
        Nombre
        +getValeur()
        +setValeur(val : double)
    }
    Reel "1" --> "1" Positif
    Reel "1" --> "1" Nombre
    Nombre "1" --> "1" Nombre
  
```

Dans l'exemple, les décorateurs sont les classes **Positif** et **Arrondi**. Pour obtenir un réel positif arrondi, on écrit juste : **new Arrondi(new Positif(new Reel(42)))**. On n'a pas eu besoin de créer la classe **RealPositifArrondi**.



## Types génériques et types paramétrés

Usage de base (4)

### Types génériques et types paramétrés

Compléments en PdO  
Aidez Degre

### Types génériques et types paramétrés

Compléments en PdO  
Aidez Degre

- À l'usage, le type générique sert de constructeur de type : on remplace le paramètre par un type concret et on obtient un **type paramétré**.

Exemple : `List<String>` est un type générique, `List<String>` un des types paramétrés que `List` permet de construire.

- Le type concret substitue le paramètre doit être un type **référence** :

`Triplet<String, String, Integer> t1 = new Triplet<String, String, Integer>("Marcel", "Durand", 23);  
Triplet<String, String, Integer> t2 = new Triplet<String, String, Integer>("Marcel", "Durand", 23);  
var t3 = new Triplet<String, String, Integer>("Marcel", "Durand", 23);`

Le type de `t1`, `t2` et `t3` est le type paramétré `Triplet<String, String, Integer>`.

Concurrence  
Interfaces graphiques

289. Pour l'instant, il semble qu'il soit prévu de permettre cela dans une prochaine version de Java.

### Parallèle entre type générique et méthode

La déclaration et l'utilisation des types génériques rappellent celles des méthodes.

#### Similitudes :

- introduction des paramètres (de type ou de valeur) dans l'entête de la déclaration ;
- utilisation des noms des paramètres dans le corps de la déclaration seulement ;
- pour utiliser le type générique ou appeler la méthode, on passe des concrétiisations des paramètres.

#### Principales différences :

- Les paramètres des génériques représentent des types alors que ceux des méthodes représentent des valeurs.
- Pour les paramètres de type, le « remplacement » 290 a lieu à la compilation.
- Pour les paramètres des méthodes, remplacement par une valeur à l'exécution.

290. Rappel : remplacement oublié, efficace, aussitôt que la vérification du bon type a été faite.

### Déclaration de paramètre de type niveau méthode

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class TroisChars extends Triplet<Char, Char, Char> {  
    public TroisChars(Char x, Char y, Char z) { super(x,y,z); }  
}
```

TroisChars étend la classe paramétrée Triplet<Char, Char, Char>.

#### Autre cas (spécialisation partielle) :

```
class DeuxCharsEtAutre extends Triplet<Char, Char, T> {  
    public DeuxCharsEtAutre(Char x, Char y, T z) { super(x,y,z); }  
}
```

La classe générique DeuxCharsEtAutre<T> étend la classe générique partiellement paramétrée Triplet<Char, Char, T>.

### Types bruts

Un nom de type générique seul, sans paramètre (comme « `Triplet<t>` »), est aussi un type légal, appelé un **type brut (raw type)**.

Son utilisation est **fortement déconseillée**, mais elle est permise pour assurer la compatibilité ascendante<sup>291</sup>.

Un type brut est supertype direct<sup>292</sup> de tout type paramétré correspondant (ex : `Triplet` est supertype direct de `Triplet<Number, Object, String>`).

Pour faciliter l'écriture, le downcast implicite de `l1` vers type paramétré compile avec l'avertissement unchecked conversion sur la deuxième ligne.

291. Un source Java <5 compile avec `javac > 5`. Or certains types sont devenus génériques, alors que d'autres ne l'étaient pas.

292. C'est une des règles de sous-type relative aux génériques, omises dans le début de ce cours.

Concurrence  
Interfaces graphiques

Remarque : il est donc, malgré tout<sup>294</sup>, possible d'écrire une méthode statique générique.

294. Rapid : les paramètres de type introduits pour la classe n'existent qu'en contexte non statique.

### Déclaration de paramètre de type niveau méthode

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class TroisChars extends Triplet<Char, Char, Char> {  
    public TroisChars(Char x, Char y, Char z) { super(x,y,z); }  
}
```

TroisChars étend la classe paramétrée Triplet<Char, Char, Char>.

#### Autre cas (spécialisation partielle) :

```
class DeuxCharsEtAutre extends Triplet<Char, Char, T> {  
    public DeuxCharsEtAutre(Char x, Char y, T z) { super(x,y,z); }  
}
```

La classe générique DeuxCharsEtAutre<T> étend la classe générique partiellement paramétrée Triplet<Char, Char, T>.

### Bornes de paramètres de type (1)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>295</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (2)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class RunnablePriorityList<T extends Comparable<T> & Runnable> {  
    ...  
}
```

« Comparable<T> & Runnable » est un **type intersection**<sup>296</sup>, il est sous-type direct de Comparable<T> et de Runnable.

Ainsi, T est sous-type de l'intersection (et donc de Comparable<T> et de Runnable).

295. Remarque : c'est le seul contexte où on peut écrire un type intersection (type non dénoté).  
Ainsi il n'est pas possible d'declarer explicitement une variable de type intersection.  
Implicitement à l'aide d'une méthode générique et du mot-clé var, cela est également possible :  
public static <T extends A & B> T intersectionFactory(...){ ... }

296. On verra dans la suite que les bornes inférieures existent aussi, mais elles ne s'appliquent qu'aux wildcards (et non aux paramètres de type).

### Bornes de paramètres de type (3)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>297</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (4)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>298</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (5)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>299</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (6)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>300</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (7)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>301</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (8)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>302</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (9)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>303</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (10)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>304</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (11)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>305</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (12)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>306</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (13)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>307</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (14)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>308</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (15)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>309</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (16)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>310</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (17)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>311</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (18)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>312</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (19)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>313</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (20)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes supérieures**<sup>314</sup> (se comportant comme supertypes du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (21)

Utilisation de classe générique par extension non générique (**spécialisation**) :

```
class Data extends Number {  
    public Data(double value) { super(value); }  
}
```

Data dérive de Number.

#### Pour limiter les concrétiisations autorisées, un paramètre de type admet des **bornes inférieures**<sup>315</sup> (se comportant comme sous-types du paramètre) :

```
class Calculator<Data extends Number> {  
    ...  
}
```

Ici, Data devra être concrétisé par un sous-type de Number : une instance de Data calculera sur la base d'un certain sous-type de Number, celui choisi à son instantiation.

### Bornes de paramètres de type (22)

Utilisation de classe générique par extension non générique (**spécialisation**) :

## Collections : pourquoi ?

Compléments en PdO

Afficher

Graphique

- Besoin :** représenter des « paquets », des « collections » d'objets similaires.
- Plusieurs genres de paquets/collections :** avec ou sans doublon, accès séquentiel ou aléatoire (= par indice), avec ou sans ordre, etc.
- Mais nombreux points communs :** peuvent contenir plusieurs éléments, possibilité d'intérieur de tester l'appartenance, l'inclusion etc.
- Pour chaque « genre » plusieurs représentations/implémentations de la structure (optimisant telle ou telle opération...).

Interfaces de `java.util` et `java.util.concurrent`

## Collections : pourquoi ?

Compléments en PdO

Afficher

Graphique

- Les collections génériques, introduites dans Java SE 5, remplacent avantageusement :
- Les tableaux.<sup>297</sup>.
- Les collections non génériques<sup>298</sup> de Java < 5, avec leurs éléments de type statique `Object`, qui il fallait caster avant usage.

Les collections justifient à elles seules l'introduction de la généricité dans Java.

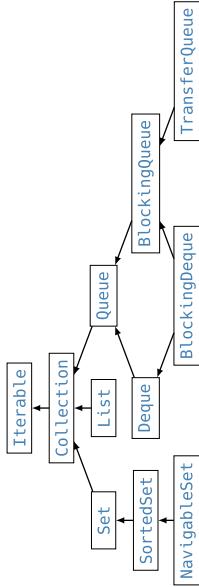
297 Qui gardent quelques avantages : syntaxe pratique, efficacité et disposent déjà d'un « genre de généricité » (un `String[]` contiendra des éléments `String` et rien d'autre), donc nous repartirons. 298. NB : les anciennes collections ont été transformées en types génériques (`Vector` <-> `Vector`) implémentant l'interface `Collection`.

Les types sans paramètre sont désormais considérés comme des types bruts et sont à éviter.

Si vous migrez du code Java < 5 vers Java ≥ 5, remplacez `ArrayList<type>` par `List<type>`.

## La hiérarchie des sous-interfaces d'Iterable

Interfaces de `java.util` et `java.util.concurrent`

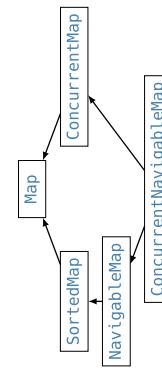


Chacune de ces interfaces possède une ou plusieurs implémentations.

299. Autres sous-interfaces dans `java.nio.file` et `java.beans.beancontext`.

## La hiérarchie des sous-interfaces de Map

Interfaces de `java.util` et `java.util.concurrent`



Chacune de ces interfaces possède une ou plusieurs implémentations.

300. Autres dans `javax.script`, `javax.xml.ws.handler.soap`.

## Itérateurs ?

Compléments en PdO

Afficher

Graphique

### Un itérateur :

- Sert à parcourir un élément et est habituellement utilisé implicitement.
- S'instancie en appelant la méthode `Iterator` sur l'objet à parcourir;
- est un objet respectant l'interface suivante :

```

public interface Iterator<E> {
    boolean hasNext();
    E next();
    void remove();
}
  
```

L'implémentation, permet de supprimer un élément en cours de parcours sans provoquer `ConcurrentModificationException` (au contraire des méthodes de l'itérable). Cette possibilité justifie de créer une variable pour manipuler explicitement l'itérateur (sinon, on préfère `for-each`).

## L'interface Iterable (2)

Interfaces de `java.util` et `java.util.concurrent`

soit en utilisant explicitement l'itérateur (rare, mais utile pour accès en écriture) :

```

public Iterator<String> it = nonIterable.iterator();
while (it.hasNext()) {
    String s = it.next();
    if (s.equals("Alement"))
        it.remove();
    else
        System.out.println("On garde " + s);
}
  
```

soit en réduisant un Stream (cf. chapitre dédié basé sur cet Iterable) :

```

    .map(Collection::stream())
    .filter(x -> !x.equals("Alement"))
    .forEach(System.out::println());
  
```

Remarque : la construction `for-each` et la méthode `foreach` ne permettent qu'un parcours en lecture seule.

soit en éduisant un Stream (cf. chapitre dédié) :

```

    .filter(x -> !x.equals("Alement"))
    .forEach(System.out::println());
  
```

Les paramètres des méthodes de Stream sont typiquement des lambdas expressions.

301. En réalité, pour des raisons assez obscures, la méthode `Stream` n'existe que dans la sous-interface Collection. Mais il est facile de programmer une méthode équivalente pour Iterable.

## L'interface Collection

Interfaces de `java.util` et `java.util.concurrent`

soit de méthodes autres que celles héritées de Collection.

Déférence : le contrat de Set garantit l'unicité de ses éléments (pas de doublon).

```

Set<Integer> s = new HashSet<Integer>();
s.add(1); s.add(2); s.add(1);
for (int i : s) System.out.println(i + " ");
  
```

Ceci affichera 1, 2, 3,

La classe HashSet est une des implementations de Set fournies par Java. C'est celle que vous utiliserez le plus souvent.

Unicité ? un élément x est unique si pour tout autre élément y, `x.equals(y)` retourne false.

⇒ importance d'avoir une redéfinition correcte de equals().

## L'interface Iterable (1)

Interfaces de `java.util` et `java.util.concurrent`

soit en utilisant explicitement l'itérateur (rare, mais utile pour accès en écriture) :

```

public interface Iterable<T> {
    Iterator<T> iterator(); // cf. Iterator
    default Spliterator<T> spliterator() { ... } // pour les Stream
    default void forEach(Consumer<T> action) { ... } // utiliser avec lambdas
}
  
```

soit avec la construction `for-each` (conseillé) :

```

for ( Object o : nonIterable ) System.out.println(o);
  
```

soit avec la méthode `forEach` et une lambda-expression (cf. chapitre dédié) :

```

nonIterable.forEach(System.out::println);
  
```

Les paramètres des méthodes de Stream sont typiquement des lambdas expressions.

301. En réalité, pour des raisons assez obscures, la méthode `Stream` n'existe que dans la sous-interface Collection. Mais il est facile de programmer une méthode équivalente pour Iterable.

## L'interface Collection

Interfaces de `java.util` et `java.util.concurrent`

soit de méthodes autres que celles héritées de Collection.

Déférence : le contrat de Set garantit l'unicité de ses éléments (pas de doublon).

```

Set<Integer> s = new HashSet<Integer>();
s.add(1); s.add(2); s.add(1);
for (int i : s) System.out.println(i + " ");
  
```

Ceci affichera 1, 2, 3,

La classe HashSet est une des implementations de Set fournies par Java. C'est celle que vous utiliserez le plus souvent.

Unicité ? un élément x est unique si pour tout autre élément y, `x.equals(y)` retourne false.

⇒ importance d'avoir une redéfinition correcte de equals().

## L'interface SortedSet

**Compteurs en POO**

Affiche Degage

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité annotations dynamiques Détournement de type Effacement de type Inversion des types classe et interface Concurrence Interfaces graphiques

**Comme Set, mais les éléments sont triés.**

... ce qui permet d'avoir quelques méthodes en plus.

```
public interface SortedSet<E> extends Set<E> {
    // Range-view
    SortedSet<E> subSet(E fromElement, E toElement);
    SortedSet<E> headset(E toElement);
    SortedSet<E> tailSet(E fromElement);

    // Endpoints
    E first();
    E last();

    // Comparator access
    Comparator<E> super E> comparator();
}
```

Implémentation typique : classe TreeSet.

## L'interface List (1)

**Compteurs en POO**

Affiche Degage

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité annotations dynamiques Détournement de type Effacement de type Inversion des types classe et interface Concurrence Interfaces graphiques

**Fonctionnalités principales :**

- acès positionnel (on peut accéder au *i*ème élément)
- recherche (si on connaît un élément, on peut demander sa position)

```
public interface List<E> extends Collection<E> {
    // Positional access
    E get(int index);
    E set(int index, E element);
    boolean add(E element); //optional
    void add(int index, E element); //optional
    E remove(int index);
    boolean addAll(int index, List<E> c); //optional
    Collection<E> subList(int from, int to);
}

// Search
int indexOf(Object o);
int lastIndexOf(Object o);
...
```

## L'interface List (1)

**Compteurs en POO**

Affiche Degage

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité annotations dynamiques Détournement de type Effacement de type Inversion des types classe et interface Concurrence Interfaces graphiques

**List** : c'est une **Collection** ordonnée avec possibilité de doublons. C'est ce qu'on utilise le plus souvent. Permet d'abstraire les notions de tableau et de liste chaînée.

**Fonctionnalités principales :**

- acès positionnel (on peut accéder au *i*ème élément)
- recherche (si on connaît un élément, on peut demander sa position)

```
public interface List<E> extends Collection<E> {
    ...
    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    ...
}

// Range-view
List<E> subList(int from, int to);
}
```

202. Vu d'un objet *E*, objet v dominant accès à une partie des données de *E*. Sans en être une copie (partielle), les modifications des 2 objets restent liées.

## Itérateurs de liste ?

**Compteurs en POO**

Affiche Degage

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité annotations dynamiques Détournement de type Effacement de type Inversion des types classe et interface Concurrence Interfaces graphiques

Un itérateur de liste sert à parcourir une liste. Il fait la même chose qu'un itérateur, mais aussi quelques autres opérations, comme :

- parcourir à l'envers
- ajouter/modifier des éléments en passant

un itérateur de liste est un objet respectant l'interface suivante :

```
public interface ListIterator<E> extends Iterator<E> {
    void add(E e);
    void addFirst(E e);
    boolean hasPrevious();
    int previousIndex();
    int nextIndex();
    void set(E e);
    void setFirst(E e);
}
```

## L'interface Queue

**Compteurs en POO**

Affiche Degage

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité annotations dynamiques Détournement de type Effacement de type Inversion des types classe et interface Concurrence Interfaces graphiques

**Queue** : c'est une **Collection** ordonnée avec possibilité de doublons. C'est ce qu'on utilise le plus souvent. Permet d'abstraire les notions de tableau et de liste chaînée.

**Fonctionnalités principales :**

- acès positionnel (on peut accéder au *i*ème élément)
- recherche (si on connaît un élément, on peut demander sa position)

```
public interface Queue<E> extends Collection<E> {
    ...
    // Iteration
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    ...
}

// Range-view
List<E> subList(int from, int to);
}
```

202. Vu d'un objet *E*, objet v dominant accès à une partie des données de *E*. Sans en être une copie (partielle), les modifications des 2 objets restent liées.

## L'interface Deque (1)

**Compteurs en POO**

Affiche Degage

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité annotations dynamiques Détournement de type Effacement de type Inversion des types classe et interface Concurrence Interfaces graphiques

**Deque** = « double ended queue ».

C'est comme une Queue, mais enrichie afin d'accéder à la collection aussi bien par le début que par la fin.

Le même Deque peut ainsi aussi bien servir de structure FIFO que LIFO (last in, first out).

```
public interface Queue<E> extends Collection<E> {
    ...
    E pollFirst();
    E pollLast();
    void push(E e);
    E pushFirst();
    E removeFirst();
    E removeLast();
    ...
}

// plus méthodes héritées

```

Implémentations typiques : ArrayDeque, LinkedList

## L'interface Map (1)

**Compteurs en POO**

Affiche Degage

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité annotations dynamiques Détournement de type Effacement de type Inversion des types classe et interface Concurrence Interfaces graphiques

**Map** est un ensemble d'associations (clé  $\rightarrow$  valeur), où chaque clé ne peut être associée qu'à une seule valeur.

Nombreuses méthodes communes avec l'interface Collection, mais particularités.

```
public interface Map<K, V> {
    ...
    V put(K key, V value);
    V removeObject(key);
    boolean containsKey(Object key);
    int size();
    boolean isEmpty();
    ...
}
```

Une Map est un ensemble d'associations (clé  $\rightarrow$  valeur), où chaque clé ne peut être associée qu'à une seule valeur.

Nombreuses méthodes communes avec l'interface Collection, mais particularités.

## L'interface Map (2)

Compléments en P00

Affiche Dégrade

Compléments en P00

Affiche Dégrade

Introduction

```
...  
// Bulk opérations  
void putAll(Map<?, extends K, ? extends V> m);  
void clear();  
// Collection View  
public Set<?> keySet();  
public Collection<?> values();  
public Set<Entry<?, ?, V> entrySet();  
// Interface for entryset elements  
public interface Entry {  
    K getKey();  
    V getValue();  
    V setValue(V value);  
}
```

Implémentation la plus courante : la classe `HashMap`

Compléments en P00

Affiche Dégrade

Introduction

**Avantages des tableaux :** syntaxe légère et efficacité.  
**Avantages des collections génériques :**

- polyvalence (plein de collections adaptées à des cas différents)
- polymorphisme via les interfaces de collections
- sûreté du type (« vraie » générativité)

**Conclusion, utilisez les collections, sauf :**

- si vous prototyppez un programme très rapidement et vous appréciez la simplicité
- si vous souhaitez optimiser la performance au maximum

304. Cela dit, les méthodes du `collection framework`, sont écrites et optimisées par des experts et déjà testées par des milliers de programmeurs. Pensez-vous faire mieux ? (peut-être, si besoin très spécifique)  
305. Mais pourquoi programmez-vous en Java alors ?

Compléments en P00

Affiche Dégrade

Introduction

**Une solution pas trop mauvaise :** 307 la classe `java.util.Optional`  
**une instance de Optional<T>** est une valeur représentant soit une instance présente de T soit l'absence d'une instance (par définition, de façon non ambiguë).  
**Optional<T>** comme un type représentant des collections de 0 ou 1 élément.

**Optional<T>** contient quelques avantages intéressants (comme la nécessité d'appeler `isPresent` et `get`). De plus, même une expression de type `Optional` est elle-même nulleable...  
307. Les valeurs nullables gardent quelques avantages intéressants (comme la nécessité d'appeler `isPresent` et `get`). Des alternatives existent (hors Java : notamment systèmes de types contenants des types non nullables, en Java : annotations `@NotNull` et `@Nullable` + outil d'analyse statique),

Compléments en P00

Affiche Dégrade

Introduction

**Pourquoi c'est plus sûr qu'un type nullable :**

- On ne peut pas appeler directement les méthodes de T sur une expression de type `Optional<T>` il faut d'abord extraire son contenu (méthode `get`).
  - Ainsi pas de risque de `NullPointerException` (ni sur l'instance d'`Optional<T>` ni sur le résultat de `get()`).
  - `get` peut bien lancer `NoSuchElementException`, mais ça se produit là où `get` est appellée. On voit donc tout de suite si et où on a oublié d'appeler `isPresent`.

**Exemple :**

```
Optional<Client> maybeRes = seat.getReservation();  
if (maybeRes.isPresent()) {  
    Client res = maybeRes.get(); // on est sûr qu'il n'y a pas d'exception  
    if (optional.isPresent()) // aucun risque de NPE car res est résultat de get()  
        res.sendReminder(); // aucun risque de NPE car res est résultat de get()  
    }  
return optional.isEmpty();  
}
```

**Remarque :** cela peut aussi s'écrire  
`seat.getReservation().ifPresent(res -> res.sendReminder());`

## L'interface SortedMap

Compléments en P00

Affiche Dégrade

Introduction

```
SortedMap est à Map ce que SortedSet est à Set : ainsi les associations sont ordonnées par rapport à l'ordre de leurs clés.  
...  
public interface SortedMap<K, V extends Map<K, V> {  
    Comparator<? super K> comparator();  
    SortedSet<K> subMap(K fromKey, K toKey);  
    SortedSet<K> headMap(K toKey);  
    SortedSet<K> tailMap(K fromKey);  
    K firstKey();  
    K lastKey();  
}
```

Implémentation typique : `TreeMap`.

Appeler une fabrique plutôt qu'un constructeur évite de choisir une implémentation : on fait confiance à la fabrique pour choisir la meilleure pour les paramètres donnés.

303. Noter le's'

Compléments en P00

Affiche Dégrade

Introduction

**Solutions (pas très bonnes) :**

- retourner une valeur qui peut être `null` (`nullable`)
  - si `getVal()` retourne une valeur nullable, l'appel `getVal().doSomething()` peut déclencher bien plus loin dans le programme (débogage difficile).
  - `null` peut aussi représenter une variable pas encore initialisée (ambiguïté)
  - lance une exception pour l'absence de valeur.
- Inconvénient : obligation d'utiliser des `try catch` (lourdeur syntaxique, s'intègre mal au flot du programme), mécanisme coûteux à l'exécution.
- Utiliser une liste à 0 ou 1 élément.
- Inconvénient : le type liste autorise les listes à 2 éléments ou plus.

306. L'invention de `null` a été qualifiée *a posteriori* par son auteur, Tony Hoare, d'`erreur à un milliard de dollars`, ce n'est pas peu dire!

Compléments en P00

Affiche Dégrade

Introduction

**Le problème :** représenter de façon non ambiguë le fait qu'une méthode puisse retourner (ou qu'une variable puisse contenir) aussi bien une valeur qu'une absence de valeur.  
**Exemples :**

- résultat du déploiement d'une pile (peut-être vide)
- recherche d'un élément satisfaisant un certain critère dans une liste (ne contenant pas forcément un tel élément)
- identité de la personne ayant réservé un certain siège dans un avion (peut-être pas encore réservé)

306. L'invention de `null` a été qualifiée *a posteriori* par son auteur, Tony Hoare, d'`erreur à un milliard de dollars`, ce n'est pas peu dire!

Compléments en P00

Affiche Dégrade

Introduction

**Les optionnels**  
**Pourquoi c'est plus sûr qu'un type nullable :**

- On ne peut pas appeler directement les méthodes de T sur une expression de type `Optional<T>`. Il faut d'abord extraire son contenu (méthode `get`).
  - Ainsi pas de risque de `NullPointerException` (ni sur l'instance d'`Optional<T>` ni sur le résultat de `get()`).
  - `get` peut bien lancer `NoSuchElementException`, mais ça se produit là où `get` est appellée. On voit donc tout de suite si et où on a oublié d'appeler `isPresent`.

**Exemple :**

```
Optional<Client> maybeRes = seat.getReservation();  
if (maybeRes.isPresent()) {  
    Client res = maybeRes.get(); // on est sûr qu'il n'y a pas d'exception  
    if (optional.isPresent()) // aucun risque de NPE car res est résultat de get()  
        res.sendReminder(); // aucun risque de NPE car res est résultat de get()  
    }  
return optional.isEmpty();  
}
```

Compléments en P00

Affiche Dégrade

Introduction

**Une solution pas trop mauvaise :** 307 la classe `java.util.Optional`  
**une instance de Optional<T>** est une valeur représentant soit une instance présente de T soit l'absence d'une instance (par définition, de façon non ambiguë).  
**Optional<T>** comme un type représentant des collections de 0 ou 1 élément.

**Optional<T>** contient quelques avantages intéressants (comme la nécessité d'appeler `isPresent` et `get`). De plus, même une expression de type `Optional` est elle-même nulleable...  
307. Les valeurs nullables gardent quelques avantages intéressants (comme la nécessité d'appeler `isPresent` et `get`). Des alternatives existent (hors Java : notamment systèmes de types contenants des types non nullables, en Java : annotations `@NotNull` et `@Nullable` + outil d'analyse statique),

Compléments en P00

Affiche Dégrade

Introduction

**Les optionnels**  
**Bien qu'Optional<T>** n'implémente pas `Collection<T>`, il est pertinent d'imaginer `Optional<T>` comme un type représentant des collections de 0 ou 1 élément.  
**Optional<T>** aussi, instance de `Optional`<`T`> contient quelques avantages intéressants (comme la nécessité d'appeler `isPresent` et `get`). De plus, même une expression de type `Optional` est elle-même nulleable...  
307. Les valeurs nullables gardent quelques avantages intéressants (comme la nécessité d'appeler `isPresent` et `get`). Des alternatives existent (hors Java : notamment systèmes de types contenants des types non nullables, en Java : annotations `@NotNull` et `@Nullable` + outil d'analyse statique),

308. Inspiration des langages fonctionnels : la classe `Option` à la monade `Maybe` en Haskell

Compléments en P00

Affiche Dégrade

Introduction

**Les optionnels**  
**Ah... et comment les instancier au fait ?**

- La classe `Optional` est munie de 2 fabricues statiques principales :
  - `Optional.of(T elem)` : si `elem` est non `null`, retourne un optional contenant `elem` (sinon `NullPointerException`)
  - `Optional.empty()` : retourne un optional vide du type désiré (en fonction du contexte)

**Exemple :**

```
public static Optional<Integer> findIndex(int[] elems, int elem) {  
    for (int i = 0; i < elems.length; i++) {  
        if (elems[i] == elem) return Optional.of(i);  
    }  
    return Optional.empty();  
}
```

Compléments en P00

Affiche Dégrade

Introduction

**Les optionnels**  
**Les optionnels**  
**Ah... et comment les instancier au fait ?**

- La classe `Optional` est munie de 2 fabricues statiques principales :

**Les optionnels**  
**Ah... et comment les instancier au fait ?**

- La classe `Optional` est munie de 2 fabricues statiques principales :

|                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h2>Fonctions de première classe et fonctions d'ordre supérieur</h2> <p>Le pionson (2)</p> | <p><b>Compléments en P00</b></p> <p>Afficher/Degager</p> <p><b>Fonctions de première classe et fonctions d'ordre supérieur</b></p> <p><b>Le besoin (1)</b></p> <p><b>Appeler 5 fois une méthode f déjà connue :</b></p> <pre>f(); f(); f(); f(); f();</pre> <p><b>Appeler f un nombre de fois inconnu à l'avance :</b></p> <pre>// Faute ! On ajoute un paramètre int :<br/>public static void repeatF(int n) { for (int i = 0; i &lt; n; i++) f(); }</pre> <p><b>Appeler 5 fois une méthode inconnue à l'avance ?</b></p> <pre>// Hm... il faudrait passer une méthode en paramètre ? Tentative :<br/>public static void repeat5(??? f) { // auquel type pour f ?<br/>for (int i = 0; i &lt; 5; i++) f(); } // si f une variable, "f/" =&gt; erreur de syntaxe<br/>}</pre> <p><b>repeat5 = fonction avec paramètre fonction = fonction d'ordre supérieur (FOS).</b></p> <ul style="list-style-type: none"> <li>Pour que cela existe, il faut des fonctions considérées comme des valeurs (passables en paramètre) par le langage : des <b>fonctions de première classe</b> (FPC).</li> </ul> <p><small>309. Voir chapitre programmation concurrente. La programmation concurrente a probablement été un argument principal pour l'introduction des lambdas-expression en Java.</small></p> <p><small>310. A comparer avec <b>while</b>(...) ... <b>for</b>(...) ... <b>switch</b>(...) ... <b>if</b> (...) ... <b>else</b> ... ....</small></p> |
| <h2>Les optionnels</h2> <p>Optionnels spécialisés</p>                                      | <p><b>Compléments en P00</b></p> <p>Afficher/Degager</p> <p><b>Fonctions de première classe et fonctions d'ordre supérieur</b></p> <p><b>Le besoin (1)</b></p> <p><b>Appeler 5 fois une méthode f déjà connue :</b></p> <pre>f(); f(); f(); f(); f();</pre> <p><b>Appeler f un nombre de fois inconnu à l'avance :</b></p> <pre>// Faute ! On ajoute un paramètre int :<br/>public static void repeatF(int n) { for (int i = 0; i &lt; n; i++) f(); }</pre> <p><b>Appeler 5 fois une méthode inconnue à l'avance ?</b></p> <pre>// Hm... il faudrait passer une méthode en paramètre ? Tentative :<br/>public static void repeat5(??? f) { // auquel type pour f ?<br/>for (int i = 0; i &lt; 5; i++) f(); } // si f une variable, "f/" =&gt; erreur de syntaxe<br/>}</pre> <p><b>repeat5 = fonction avec paramètre fonction = fonction d'ordre supérieur (FOS).</b></p> <ul style="list-style-type: none"> <li>Pour que cela existe, il faut des fonctions considérées comme des valeurs (passables en paramètre) par le langage : des <b>fonctions de première classe</b> (FPC).</li> </ul>                                                                                                                                                                                                                                                                                                                                                    |
| <h2>Les optionnels</h2> <p>Optionnels spécialisés</p>                                      | <p><b>Compléments en P00</b></p> <p>Afficher/Degager</p> <p><b>Fonctions de première classe et fonctions d'ordre supérieur</b></p> <p><b>Le besoin (1)</b></p> <p><b>Appeler 5 fois une méthode f déjà connue :</b></p> <pre>f(); f(); f(); f(); f();</pre> <p><b>Appeler f un nombre de fois inconnu à l'avance :</b></p> <pre>// Faute ! On ajoute un paramètre int :<br/>public static void repeatF(int n) { for (int i = 0; i &lt; n; i++) f(); }</pre> <p><b>Appeler 5 fois une méthode inconnue à l'avance ?</b></p> <pre>// Hm... il faudrait passer une méthode en paramètre ? Tentative :<br/>public static void repeat5(??? f) { // auquel type pour f ?<br/>for (int i = 0; i &lt; 5; i++) f(); } // si f une variable, "f/" =&gt; erreur de syntaxe<br/>}</pre> <p><b>repeat5 = fonction avec paramètre fonction = fonction d'ordre supérieur (FOS).</b></p> <ul style="list-style-type: none"> <li>Pour que cela existe, il faut des fonctions considérées comme des valeurs (passables en paramètre) par le langage : des <b>fonctions de première classe</b> (FPC).</li> </ul>                                                                                                                                                                                                                                                                                                                                                    |
| <h2>Fonctions de première classe et fonctions d'ordre supérieur</h2> <p>Le pionson (2)</p> | <p><b>Compléments en P00</b></p> <p>Afficher/Degager</p> <p><b>Fonctions de première classe et fonctions d'ordre supérieur</b></p> <p><b>Le besoin (3)</b></p> <p><b>Encore un exemple : 311</b></p> <pre>// toujours en syntaxe fantaisiste --- SURTOUT NE PAS RECOPIER OU N'ÉCRIT RETENIR ! public static &lt;U&gt; V&gt; List&lt;&gt; mapList(&lt;U&gt; 1, ??; f) {     List&lt;&gt; ret = new ArrayList&lt;&gt;();     for (U x : 1) ret.add(f(x));     return ret; }</pre> <p><b>Ou encore : 312</b></p> <pre>public static &lt;U&gt; ReadPacket&lt;Societ&lt;S, ??&gt; callback) {     ... }</pre> <p><b>callback = FPC pour traiter le prochain paquet reçu = fonction de rappel/callback.</b></p> <p><small>311. L'API java.util.stream content plein de méthodes de traitement par lot dans ce genre.</small></p> <p><small>312. Lecture asynchrone, similaire à ce qu'on trouve dans l'API java.nio.</small></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |

## FPC avant Java 8

## FPC avant Java 8

## FPC avant Java 8

|                    |                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | Bilan                                                                                                                                                                                                                                                                     |
| Introduction       | Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et annotations Classes et interfaces Généralités Les expressions Effacement de type Utilisation des méthodes génériques Classes et interfaces Concurrence Interfaces graphiques |
| Compléments en P00 | Bilan                                                                                                                                                                                                                                                                     |

### Bilan pour les FPC avant Java 8 :

- **typage** : au cas par cas, rien de prévu, pas de standard ; chaque méthode peut spécifier une interface différente pour la fonction passée en argument.  
→ à partir de Java 8 : le package `java.util.function` propose une série d'**interfaces fonctionnelles** standard.
- **syntaxe** : lourde et peu pratique.  
→ à partir de Java 8, on peut écrire des **lambda-expressions**.
- **représentation en mémoire** : instance d'une classe contenant juste une méthode.  
→ comme c'est une idée raisonnable, ça ne change pas.

### Bilan pour les FPC avant Java 8 :

- **typage** : au cas par cas, rien de prévu, pas de standard ; chaque méthode peut spécifier une interface différente pour la fonction passée en argument.
- **syntaxe** : lourde et peu pratique.  
→ à partir de Java 8, on peut écrire des **lambda-expressions**.
- **représentation en mémoire** : instance d'une classe contenant juste une méthode.

### Bilan pour les FPC avant Java 8 :

- **typage** : au cas par cas, rien de prévu, pas de standard ; chaque méthode peut spécifier une interface différente pour la fonction passée en argument.  
→ à partir de Java 8 : le package `java.util.function` propose une série d'**interfaces fonctionnelles** standard.
- **Sinon**, rien n'est changé au système de type de Java (même sa syntaxe).
- **syntaxe** : lourde et peu pratique.  
→ à partir de Java 8, on peut écrire des **lambda-expressions**.
- **représentation en mémoire** : instance d'une classe contenant juste une méthode.

### Bilan pour les FPC avant Java 8 :

- **typage** : au cas par cas, rien de prévu, pas de standard ; chaque méthode peut spécifier une interface différente pour la fonction passée en argument.  
→ à partir de Java 8 : le package `java.util.function` propose une série d'**interfaces fonctionnelles** standard.
- **Sinon**, rien n'est changé au système de type de Java (même sa syntaxe).
- **syntaxe** : lourde et peu pratique.  
→ à partir de Java 8, on peut écrire des **lambda-expressions**.
- **représentation en mémoire** : instance d'une classe contenant juste une méthode.

## Bilan

## Bilan

## Bilan

## FPC avant Java 8

## FPC avant Java 8

## FPC avant Java 8

|                    |                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | Bilan                                                                                                                                                                                                                                                                     |
| Introduction       | Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et annotations Classes et interfaces Généralités Les expressions Effacement de type Utilisation des méthodes génériques Classes et interfaces Concurrence Interfaces graphiques |
| Compléments en P00 | Bilan                                                                                                                                                                                                                                                                     |

### Bilan pour les FPC avant Java 8 :

- **typage** : au cas par cas, rien de prévu, pas de standard ; chaque méthode peut spécifier une interface différente pour la fonction passée en argument.  
→ à partir de Java 8 : le package `java.util.function` reste le standard pour les « fonctions » 317 de cette série d'**interfaces fonctionnelles** standard.
- **Sinon**, rien n'est changé au système de type de Java (même sa syntaxe).
- **syntaxe** : lourde et peu pratique.  
→ à partir de Java 8, on peut écrire des **lambda-expressions**.
- **représentation en mémoire** : instance d'une classe contenant juste une méthode.  
→ comme c'est une idée raisonnable, ça ne change pas.

### Bilan pour les FPC avant Java 8 :

- **typage** : au cas par cas, rien de prévu, pas de standard ; chaque méthode peut spécifier une interface différente pour la fonction passée en argument.  
→ à partir de Java 8 : le package `java.util.function` reste le standard pour les « fonctions » 317 de cette série d'**interfaces fonctionnelles** standard.
- **Sinon**, rien n'est changé au système de type de Java (même sa syntaxe).
- **syntaxe** : lourde et peu pratique.  
→ à partir de Java 8, on peut écrire des **lambda-expressions**.
- **représentation en mémoire** : instance d'une classe contenant juste une méthode.

### Bilan pour les FPC avant Java 8 :

- **typage** : au cas par cas, rien de prévu, pas de standard ; chaque méthode peut spécifier une interface différente pour la fonction passée en argument.  
→ à partir de Java 8 : le package `java.util.function` propose une série d'**interfaces fonctionnelles** standard.
- **Sinon**, rien n'est changé au système de type de Java (même sa syntaxe).
- **syntaxe** : lourde et peu pratique.  
→ à partir de Java 8, on peut écrire des **lambda-expressions**.
- **représentation en mémoire** : instance d'une classe contenant juste une méthode.

### Bilan pour les FPC avant Java 8 :

- **typage** : au cas par cas, rien de prévu, pas de standard ; chaque méthode peut spécifier une interface différente pour la fonction passée en argument.  
→ à partir de Java 8 : le package `java.util.function` propose une série d'**interfaces fonctionnelles** standard.
- **Sinon**, rien n'est changé au système de type de Java (même sa syntaxe).
- **syntaxe** : lourde et peu pratique.  
→ à partir de Java 8, on peut écrire des **lambda-expressions**.
- **représentation en mémoire** : instance d'une classe contenant juste une méthode.

## Catalogue des interfaces fonctionnelles de Java 8

## Catalogue des interfaces fonctionnelles de Java 8

## Catalogue des interfaces fonctionnelles de Java 8

|                    |                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | Bilan                                                                                                                                                                                                                                                                     |
| Introduction       | Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et annotations Classes et interfaces Généralités Les expressions Effacement de type Utilisation des méthodes génériques Classes et interfaces Concurrence Interfaces graphiques |
| Compléments en P00 | Bilan                                                                                                                                                                                                                                                                     |

## Catalogue des interfaces fonctionnelles de Java 8

## Catalogue des interfaces fonctionnelles de Java 8

## Catalogue des interfaces fonctionnelles de Java 8

|                    |                                                                                                                                                                                                                                                                           |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | Bilan                                                                                                                                                                                                                                                                     |
| Introduction       | Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et annotations Classes et interfaces Généralités Les expressions Effacement de type Utilisation des méthodes génériques Classes et interfaces Concurrence Interfaces graphiques |
| Compléments en P00 | Bilan                                                                                                                                                                                                                                                                     |

### Interfaces spécialisées :

- **Type représenté** | Méthode unique  
`boolean getAsBoolean()`  
`double applyAsDouble(double, double)`  
`void accept(double)`  
`R apply(double)`  
`void test(double)`  
`double getAsDouble()`  
`double applyAsInt(double)`  
`int applyAsInt(double)`  
...

Cf. javadoc de `java.util.function` pour liste complète.

Intérêt des interfaces spécialisées : programmes mieux optimisés 316 qui avec les types « emballés » (`Int`, `Long`, ...).

316. Moins d'allocations et d'indirections.

L'annotation facultative `@FunctionalInterface` demande au compilateur de signaler une erreur si ce qui suit n'est pas une définition d'interface fonctionnelle.

317. Ces fonctions sont intéressantes pour leurs effets de bord et non pour la transformation qu'elles représentent. En effet, en mathématiques, `card({}) → {()}` = 1.

public static void repeat5(Runnable f) { for (int i = 0; i < 5; i++) f.run(); }

```
public static void tFE5(BoltSupplier cord, Runnable tBlock, Runnable tBlock, Runnable tBlock, Runnable tBlock, Runnable tBlock) {
    if (cond.getOrDefault(true))
        else elseBlock.run();
}
```

public static void retry(Runnable instructions, int tries) {
 while (tries > 0) {
 try {
 instructions.run();
 } catch (Throwable t) {
 if (tries-- > 0)
 t.printStackTrace();
 else throw new RunTimeException("Failure\_persisted\_after\_all\_tries.");
 }
}

```
public static void map(List<?> l, Function<?,?> f) {
    List<?> ret = new ArrayList<?>();
    for (Object x : l) ret.add(f.apply(x));
    return ret;
}
```

... avec les bons types et la bonne syntaxe

Retour sur les exemples...  
dans le package `java.util.function` mais pas seulement

... avec les bons types et la bonne syntaxe

public static void map(List<?> l, Function<?,?> f) {

List<?> ret = new ArrayList<?>();
 for (Object x : l) ret.add(f.apply(x));
 return ret;
}

Retour sur les exemples...  
dans le package `java.util.function` mais pas seulement

... avec les bons types et la bonne syntaxe

## Interfaces fonctionnelles

Compléments en POO  
Aidez Degorre

Écrire les types des FPC, c'est bien. Mais comment exécuter une fonction ?

- Comme on a déjà pu voir sur les exemples :
- Il n'y a pas de syntaxe réservée pour exécuter une expression fonctionnelle.
  - Il faut donc à chaque fois appeler explicitement la méthode de l'interface fonctionnelle concernée (et donc connaître son nom...).

Exemple :

```
Function<Integer, Integer> carre = n -> n * n;
System.out.println(carre.apply(5)); // <-- ici c'est apply mais...
Predicate<Integer> estPair = n -> (n % 2) == 0;
System.out.println(estPair.test(5)); // <-- là c'est test
```

Interface graphique

## Syntaxe des lambda-expressions

Compléments en POO  
Aidez Degorre

lambda-abstraction : syntaxe formelle

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généralité Classes et interfaces abstraites et concrètes Classes statiques Généralité Classes et interfaces abstraites et concrètes Classes statiques Compatibilité Classes et interfaces abstraites et concrètes Classes statiques Interfaces graphiques

- Écrire les types des FPC, c'est bien. Mais comment exécuter une fonction ?

- Comme on a déjà pu voir sur les exemples :
- Il n'y a pas de syntaxe réservée pour exécuter une expression fonctionnelle.
  - Il faut donc à chaque fois appeler explicitement la méthode de l'interface fonctionnelle concernée (et donc connaître son nom...).
- Exemple :
- ```
<corps de la fonction>
x -> x + 2
(raccourci pour (int x)-> { return x + 2; })
```
- Exemples :
- `x -> x + y`
  - `(raccourci pour (int x, int y)-> { return x + y; })`
  - `(a, b) -> {
 int q = 0;
 while (a >= b) { q++; a -= b; }
 return q;
}`

Écrire une fonction anonyme par lambda-abstraction :

- ```
<paramètres> -> <corps de la fonction>
<paramètres> . liste de paramètres formels (de 0 à plusieurs), de la forme
(int x, int y, String s)
Mais juste (x, y, s) fonctionne aussi (type des paramètres inféré).
Et parenthèses facultatives quand il y a un seul paramètre.
Il est aussi possible (Java ≥ 11) de remplacer les noms de types par var.
<corps de la fonction>, au choix :
```

- une simple expression, p. ex. `(x==?5:""`
- une liste d'instructions entre accolades, contenant une instruction `return` si type de retour non **void**

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généralité Classes et interfaces abstraites et concrètes Classes statiques Généralité Classes et interfaces abstraites et concrètes Classes statiques Compatibilité Classes et interfaces abstraites et concrètes Classes statiques Interfaces graphiques

## Syntaxe des lambda-expressions

Compléments en POO  
Aidez Degorre

lambda-abstraction : par exemple

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généralité Classes et interfaces abstraites et concrètes Classes statiques Généralité Classes et interfaces abstraites et concrètes Classes statiques Compatibilité Classes et interfaces abstraites et concrètes Classes statiques Interfaces graphiques

- Écrire une fonction anonyme par lambda-abstraction :

- ```
<paramètres> -> <corps de la fonction>
<paramètres> . liste de paramètres formels (de 0 à plusieurs), de la forme
(int x, int y, String s)
```

- Mais juste (x, y, s) fonctionne aussi (type des paramètres inféré).  
Et parenthèses facultatives quand il y a un seul paramètre.  
Il est aussi possible (Java ≥ 11) de remplacer les noms de types par var.  
`<corps de la fonction>`, au choix :

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généralité Classes et interfaces abstraites et concrètes Classes statiques Généralité Classes et interfaces abstraites et concrètes Classes statiques Compatibilité Classes et interfaces abstraites et concrètes Classes statiques Interfaces graphiques

## Syntaxe des lambda-expressions

Compléments en POO  
Aidez Degorre

Référence de méthode : les cas de figure

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généralité Classes et interfaces abstraites et concrètes Classes statiques Généralité Classes et interfaces abstraites et concrètes Classes statiques Compatibilité Classes et interfaces abstraites et concrètes Classes statiques Interfaces graphiques

- Supposons la classe suivante définie :
- ```
class C {
    int val;
    int val() { this.val = val; }
    static int int() { int n = return n; }
    int gint(n) { return val + n; }
```

La notation « référence de méthode » se décline pour différents cas de figure :

- Méthode statique → `C::f pour n -> C.f(n)`
- Méthode d'instance avec récepteur donné → avec `x = new C()`, on écrit `x::g`
- Calculer les racines carrées des nombres d'une liste → `List<Double> racines = map(maListe, Math::sqrt);`

- En cas de surcharge, Java déduit la méthode référencée du type attendu.

## Ce qui se cache derrière les lambda-expressions

Compléments en POO  
Aidez Degorre

Concernant le type (1) – petits piégés

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généralité Classes et interfaces abstraites et concrètes Classes statiques Généralité Classes et interfaces abstraites et concrètes Classes statiques Compatibilité Classes et interfaces abstraites et concrètes Classes statiques Interfaces graphiques

- Ce qui se cache derrière les lambda-expressions

Concernant le type (2) – petits piégés

Hors contexte, une lambda-expression n'a pas de type (plusieurs types possibles).

- En contexte, sous réserve de compatibilité, son type est le type attendu à son emplacement dans le programme (**inférence de type**).

- Compatibilité si :
- le type attendu est défini par une interface fonctionnelle (= est un type SAM) et la méthode abstraite de cette interface est redéfinissable par une méthode qui aurait la même signature et le même type de retour que la lambda-expression.

- Exemple, on peut écrire `Function<Integer, Double> f = x -> Math.sqrt(x);` car l'interface `Function` est comme suit :

```
public interface Function<T, P> {
    R apply(T t);
}
```

Oublier, dans le cas où les types des arguments ne sont pas précis, s'il existe une façon de les ajouter qui rend la signature compatible.

## Syntaxe des lambda-expressions

Compléments en POO  
Aidez Degorre

Référence de méthode

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généralité Classes et interfaces abstraites et concrètes Classes statiques Généralité Classes et interfaces abstraites et concrètes Classes statiques Compatibilité Classes et interfaces abstraites et concrètes Classes statiques Interfaces graphiques

- Pour créer une lambda-expression contenant juste l'appel d'une méthode existante :
- on peut utiliser la lambda-abstraction : `x -> Math.sqrt(x)`
  - mais il existe une notation encore plus compacte :

`Math::sqrt`

Ceci s'appelle une **référence de méthode**

Remarque :

- `Math::sqrt` est bien équivalent à `x -> Math.sqrt(x)`, et non à `(double x)-> Math.sqrt(x)`. Cela a une incidence pour l'inférence de type (cf. la suite).
- Exemples de situations où cela devient intéressant :
- À l'exécution, comment est évaluée une lambda-expression ?
  - À la compilation, étant donnée une lambda-expression, quel type le compilateur lui donne-t-il ?

## Ce qui se cache derrière les lambda-expressions

Compléments en POO  
Aidez Degorre

Concernant le type (1)

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généralité Classes et interfaces abstraites et concrètes Classes statiques Généralité Classes et interfaces abstraites et concrètes Classes statiques Compatibilité Classes et interfaces abstraites et concrètes Classes statiques Interfaces graphiques

Ce qui se cache derrière les lambda-expressions

Concernant le type (2) – petits piégés

Le fait de préciser le type des paramètres d'une lambda-expression restreint les possibilités d'utilisation.

- Ces exemples complient :

- ```
Function<Integer, Double> f = (double x) -> Math.sqrt(x);
Function<Integer, Double> f = (integer x) -> Math.sqrt(x);
Function<Double, Double> f = (double x) -> Math.sqrt(x); // pourtant la lambda-expression accepte double (plus large que int).
```

Mais ceux-ci ne complient pas :

- ```
Function<Integer, Double> f = (Double x) -> Math.sqrt(x);
Function<Double, Double> f = (integer x) -> Math.sqrt(x); // pourtant la lambda-expression accepte double (plus large que int).
```

Remarquablement, ceci compile (malgré le fait que `sqrt` ait un paramètre **double**) :

```
Function<Integer, Double> f = Math::sqrt;
```

## Avertissement

Attention aux variables locales !

**Attention :** n'importe quel type SAM peut être le type d'une lambda-expression. Pas seulement ceux définis dans `Java.util.function`.

Pourtout où une expression de type SAM attendue, on peut utiliser une lambda-expression, même si ce type (ou la méthode qui l'attend) date d'avant Java 8.

Ainsi, en **Swing**, à la place de la classique « invocation magique » :

```
SwingUtilities.invokeLater( new Runnable() { MontG_butUdf(); } );
```

on peut écrire :  `SwingUtilities.invokeLater(( ()-> MontIG.build() );`

ou encore mieux :  `SwingUtilities.invokeLater(MonIG::build);`

319. En fait, pour les lambdas, la JVM construit la classe anonyme à l'exécution seulement. À cet effet, Java a en fait compilé l'expression en écriturant l'instruction `invokedynamic`. (introduite dans Java 9 dans ce but).

Autrement, les classes, même anonymes, sont créées à la compilation et existent déjà dans le code octet.

## Ce qui se cache derrière les lambda-expressions

Concernant le type (3)

Compléments en P00

Affiche/Dégrade

Introduction  
Généralités

Style

Objets et classes

Généralité

Objets et classes

Concurrence

Interfaces graphiques

Types et polymorphisme

Héritage

Généralité

Objets et classes

Concurrence

Interfaces graphiques

Lambda-expressions

Effacement de type

Éffacement des attributs

Éffacement des méthodes

Éffacement des champs

Éffacement des constructeurs

Éffacement des méthodes statiques

Éffacement des champs statiques

Éffacement des constructeurs statiques

Éffacement des méthodes privées

Éffacement des champs privés

Éffacement des méthodes publiques

Éffacement des champs publiques

Éffacement des méthodes protected

Éffacement des champs protected

Éffacement des méthodes final

Éffacement des champs final

Éffacement des méthodes static

Éffacement des champs static

Éffacement des méthodes abstract

Éffacement des champs abstract

Éffacement des méthodes interface

Éffacement des champs interface

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

## Avertissement

Concernant le type (3)

Compléments en P00

Affiche/Dégrade

Introduction

Généralités

Style

Objets et classes

Généralité

Objets et classes

Concurrence

Interfaces graphiques

Type et polymorphisme

Héritage

Généralité

Objets et classes

Concurrence

Interfaces graphiques

Lambda-expressions

Effacement de type

Éffacement des attributs

Éffacement des méthodes

Éffacement des champs

Éffacement des constructeurs

Éffacement des méthodes statiques

Éffacement des champs statiques

Éffacement des méthodes protected

Éffacement des champs protected

Éffacement des méthodes final

Éffacement des champs final

Éffacement des méthodes abstract

Éffacement des champs abstract

Éffacement des méthodes interface

Éffacement des champs interface

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

322. si un : on troque la syntaxe des lambda-expressions contre celle, plus verbale, des classes anonymes

Cette dernière technique n'a pas d'inconvénient<sup>322</sup>. C'est donc celle qu'il faut privilier.

322. si un : on troque la syntaxe des lambda-expressions contre celle, plus verbale, des classes anonymes

Compléments en P00

Affiche/Dégrade

Introduction

Généralités

Style

Objets et classes

Généralité

Objets et classes

Concurrence

Interfaces graphiques

Lambda-expressions

Effacement de type

Éffacement des attributs

Éffacement des méthodes

Éffacement des champs

Éffacement des constructeurs

Éffacement des méthodes statiques

Éffacement des champs statiques

Éffacement des méthodes protected

Éffacement des champs protected

Éffacement des méthodes final

Éffacement des champs final

Éffacement des méthodes abstract

Éffacement des champs abstract

Éffacement des méthodes interface

Éffacement des champs interface

Éffacement des méthodes overridable

Éffacement des champs overridable

Éffacement des méthodes finalizable

Éffacement des champs finalizable

Éffacement des méthodes synchronized

Éffacement des champs synchronized

Éffacement des méthodes volatile

Éffacement des champs volatile

Éffacement des méthodes transient

Éffacement des champs transient

Éffacement des méthodes native

Éffacement des champs native

Éffacement des méthodes overridable

Éffacement des champs overridable

## Opérations d'agrégation

## Opérations d'agrégation

## Opérations d'agrégation

Définition et quelques exemples

Quelques exemples (2)

**Opération d'agrégation :** traitement d'une séquence de données de même type qui produit un résultat synthétique 327 dépendant de toutes ces données.

**Exemples :**

- calcul de la taille d'une collection
  - concaténation des chaînes d'une liste de chaînes
  - transformation d'une liste de chaînes en la liste de ses longueurs (ex : "bonjour", "l'e", monde → 7, 2, 5)
  - recherche d'un élément satisfaisant un certain critère
- Tous ces calculs pourraient s'écrire à l'aide de boucles **for** très similaires...

327. synthèse = résumé

## Opérations d'agrégation

## Opérations d'agrégation

## Opérations d'agrégation

Calcul de la taille d'une collection :

```
public static int size(Collection<?> dataSource) {
    int acc = 0;
    for (Object e: dataSource) acc++;
    return acc;
}
```

Concaténation des chaînes d'une liste de chaînes :

```
public static String concat(List<String> dataSource) {
    String acc = "";
    for (String e: dataSource) acc += e.toString();
    return acc;
}
```

Recherche d'un élément satisfaisant un certain critère 328 :

Voyez-vous le motif commun ?

328. Remarque : on peut optimiser cette boucle, mais cette présentation illustre mieux le propos.

## Opérations d'agrégation

## Opérations d'agrégation

## Opérations d'agrégation

Transformation d'une liste de chaînes en la liste de ses longueurs :

```
public static List<Integer> lengths(Collection<String> dataSource) {
    List<Integer> lst = new ArrayList<Integer>();
    for (String e: dataSource) lst.add(e.length());
    return lst;
}
```

Recherche d'un élément satisfaisant un certain critère 328 :

```
public static E find(List<E> dataSource, Predicate<E> criterion) {
    E acc = null;
    for (E e: dataSource) acc = (criterion.test(e)?e:null);
    return acc;
}
```

Voyez-vous le motif commun ?

328. Remarque : on peut optimiser cette boucle, mais cette présentation illustre mieux le propos.

## Opérations d'agrégation

## Opérations d'agrégation

## Opérations d'agrégation

On garde ce qui est commun dans une méthode prenant en argument ce qui est différent :

```
public static <E, R> R fold(Iterable<E> dataSource, R zero, ??? op) {
    R acc = zero;
    for (E e : dataSource) acc = op(acc, e); // comment on écrit ça déjà ?
    return acc;
}
```

... et on se rappelle le cours sur les fonctions de première classe (FPC) et les fonctions d'ordre supérieur (FOS) :

```
public static <E, R> R fold(Iterable<E> dataSource, R zero, BiFunction<R, E, R> op) {
    R acc = zero;
    for (E e : dataSource) acc = op.apply(acc, e);
    return acc;
}
```

## Opérations d'agrégation

## Opérations d'agrégation

## Opérations d'agrégation

On garde ce qui est commun dans une méthode prenant en argument ce qui est différent :

```
public static <E, R> R fold(Iterable<E> dataSource, R zero, ??? op) {
    R acc = zero;
    for (E e : dataSource) acc = op(acc, e); // comment on écrit ça déjà ?
    return acc;
}
```

Pour écrire des traitements similaires à ces exemples, on aimerait une API fournissant des FOS analogues à **fold** pour les principaux schémas d'itération 329 .

C'est justement ce que fait l'API **stream**.

329. Similaires aux fonctions de manipulation de liste en Ocaml.

330. Mais pas seulement...

## Opérations d'agrégation

## Opérations d'agrégation

## Opérations d'agrégation

Streams : API introduite dans Java 8 pour effectuer des opérations d'agrégation.

- API dans le style fonctionnel, avec fonctions d'ordre supérieur;
- distincte de l'API des collections (nouvelle interface Stream, au lieu de méthodes ajoutées à Collection 331);
- optimisée pour les grands volumes de données : **évaluation paresseuse** (calcul effectué seulement au dernier moment, seulement lorsqu'ils sont nécessaires);
- qui sait utiliser les CPU multi-cœur pour accélérer ses calculs (implémentation parallèle multi-threadée).

Avertissement : ce chapitre traite du package java.util.stream introduit dans Java 8. Ces streams n'ont aucun rapport avec les classes InputStream et OutputStream de java.io.

331. Heureusement on obtient facilement une instance de Stream depuis une instance de Collection grâce à la méthode stream de Collection.

## Opérations d'agrégation

## Opérations d'agrégation

## Opérations d'agrégation

Pour écrire des traitements similaires à ces exemples, on aimerait une API fournissant des FOS analogues à **fold** pour les principaux schémas d'itération 329 .

```
public static <T> T fold(Iterable<T> iterable, T zero, BiFunction<T, T, T> op) {
    T acc = zero;
    for (T e : iterable) acc = op.apply(acc, e);
    return acc;
}
```

Pour écrire des traitements similaires à ces exemples, on aimerait une API fournissant des FOS analogues à **fold** pour les principaux schémas d'itération 329 .

```
public static <T> T fold(Iterable<T> iterable, T zero, BiFunction<T, T, T> op) {
    T acc = zero;
    for (T e : iterable) acc = op.apply(acc, e);
    return acc;
}
```

C'est justement ce que fait l'API **stream**.

329. Similaires aux fonctions de manipulation de liste en Ocaml.

330. Mais pas seulement...

## Opérations d'agrégation

## Opérations d'agrégation

## Opérations d'agrégation

Les calculs sont effectués à l'appel de l'opération terminale seulement. Et seuls les calculs nécessaires le sont.

332. Souvent une collection, mais peut aussi être un tableau, une fonction productrice d'éléments, un canal d'entrées/sorties

|                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h2>Les objets stream</h2> <p>Stream versus Iterator</p>    | <h2>Les objets stream</h2> <p>Compléments en PdO<br/>Aldric Degorre</p> <p>Qu'est-ce qu'un <b>stream</b> ? → <b>2 points de vue</b> :</p> <ol style="list-style-type: none"> <li>1 la représentation implicite d'une <u>séquence</u> d'éléments finie ou infinie</li> <li>2 la description d'une suite d'opérations permettant d'obtenir cette séquence.</li> </ol> <p><b>Remarques importantes :</b></p> <ul style="list-style-type: none"> <li>• Un <b>objet stream</b> n'est pas une collection : il ne contient qu'une référence vers une source d'éléments (parfois une collection, souvent un autre <b>stream</b>) et la <u>description</u> d'une opération à effectuer.</li> <li>• Un <b>objet stream</b> n'est pas le résultat d'un calcul, mais la <u>description d'un calcul à effectuer</u>.</li> </ul> <p>332. Pour les fans de programmation fonctionnelle : le type <code>Stream&lt;T&gt;</code> muni des opérations <code>of</code> et <code>f</code> (<code>f</code> rappel est une monade).</p>                                                                   |
| <h2>Streams et parallélisme</h2> <p>Problème à résoudre</p> | <h2>Streams et parallélisme</h2> <p>Problème à résoudre</p> <p>15 nombres entiers aléatoires positifs inférieurs à 100 en ordre croissant :</p> <pre>Stream&lt;Integer&gt; stream = Stream.of(1, 2, 3, ...).limit(100).sorted().map(x -&gt; x * x).collect(Collectors.toList());</pre> <p>Nombre d'usagers d'une bibliothèque ayant emprunté un livre d'Alexandre Dumas :</p> <pre>bibli.getLivre("Dumas").stream() // on obtient un Stream&lt;Double&gt; .filter(livre -&gt; livre.getAuteur() == "Dumas") // Stream&lt;Double&gt; .distinct() // Stream&lt;Double&gt; .collect(Collectors.toList()); // Stream&lt;Integer&gt; long</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <h2>Streams et parallélisme</h2> <p>Quelques exemples</p>   | <h2>Streams et parallélisme</h2> <p>Quelques exemples</p> <p>... et bien justement :</p> <p>Java permet de lancer les opérations d'agrégation en parallèle<sup>335</sup>, sans presque rien changer à l'invocation du même traitement en séquentiel :</p> <ul style="list-style-type: none"> <li>• Il suffit de créer le stream avec <code>maCollection.parallelStream()</code> à la place de <code>maCollection.stream()</code>.</li> </ul> <p>Un stream est soit (entièrement) parallèle, soit (entièrement) séquentiel. L'opération terminale prend seulement en compte le dernier appel à <code>sequential</code> ou <code>parallel</code><sup>336</sup>.</p> <p>335. En utilisant (de façon cachée) <code> ForkJoinPool.ForkJoinTask</code>. Sauf mention contraire, le <code>thread pool</code> par défaut (<code>ForkJoinPool.commonPool()</code>) est utilisé.</p> <p>336. Rappel : l'effectuation des calculs étant seulement déclenchée par l'opération terminale, il est logique que ses modalités concrètes d'exécution ne soient prises en compte qu'à ce moment.</p> |
| <h2>Streams et parallélisme</h2> <p>Problème à résoudre</p> | <h2>Streams et parallélisme</h2> <p>Problème à résoudre</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

## Appendice : les méthodes de l'API stream

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Les transparents qui suivent :**

- sont un résumé des méthodes proposées dans l'API stream.
- ne sont pas détaillés en cours magistral
- doivent servir de référence pour les TP et pour la relecture approfondie du cours.

**340. Rappel : oui, c'est possible depuis Java 8.**

340. Rappel : oui, c'est possible depuis Java 8.

## L'interface Stream

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**public interface Stream<? T> Stream<? T> { // pour des éléments de type T ... }**

**(Il existe aussi DoubleStream, IntStream et LongStream.)**

Cette interface contient un grand nombre de méthodes : 3 catégories :

- des méthodes statiques 340 servant à créer des streams depuis des sources diverses.
- des méthodes d'instance transformant des streams en streams (pour les opérations intermédiaires)
- des méthodes d'instance transformant des streams en autre chose (pour les opérations terminales).

340. Rappel : oui, c'est possible depuis Java 8.

## Fabriquer un stream depuis une source

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

- Depuis une collection : méthode Stream<? T> Stream() de la collection<? T>.
- À l'aide d'une des méthodes statiques de Stream :
  - <T> Stream<? T> empty() : retourne un stream vide
  - <T> Stream<? T> generateSupplier<? T> s) : retourne la séquence des éléments générés par s. get() (Rappel : Supplier<? T> : fonction de () {}).
  - <T> Stream<? T> iterate(T seed, UnaryOperator<? T> f) : retourne la séquence des éléments seed, f.apply(seed), f.apply(f.apply(seed)) ...
  - <T> Stream<? T> of(... values) : retourne le stream constitué de la liste des éléments passés en argument (méthode d'arité variable).
- En utilisant un builder<sup>341</sup> (Stream.Builder) :
  - Un Stream.Builder est un objet mutable servant à construire un stream.
  - On instancie un builder vide avec l'appel statique b = Stream.builder()
  - On ajoute des éléments avec les appels b.add(T e) ou b.accept(T e).
  - On finalise en créant le stream contenant ces éléments : appel s = b.build()

341. On parle du patron de conception builder (ou "moniteur"), ici appliquée aux streams. Ainsi, par exemple, il existe une classe StringBuilder jouant le même rôle pour les String. Voir le FP sur le patron builder.

## L'interface Stream

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**public interface Stream<? T> Stream<? T> { // pour des éléments de type T ... }**

**(Il existe aussi DoubleStream, IntStream et LongStream.)**

Cette interface contient un grand nombre de méthodes : 3 catégories :

- des méthodes statiques 340 servant à créer des streams depuis des sources diverses.
- des méthodes d'instance transformant des streams en streams (pour les opérations intermédiaires)
- des méthodes d'instance transformant des streams en autre chose (pour les opérations terminales).

340. Rappel : oui, c'est possible depuis Java 8.

## Calculer et extraire un résultat : les opérations terminales

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Transformer un stream : les opérations intermédiaires (2)**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Calculer et extraire un résultat : les opérations terminales (1)**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Transformer un stream : les opérations intermédiaires (1)**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Pour this instance de Stream<? T> :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**retourne un stream qui parcourt les éléments de this sans les doublons.**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Stream<? T> filter(Predicate<? T> p) :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**retourne le stream parcourant les éléments de this qui satisfisent p.test(x)**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**<? T> flatMap(Function<? T, ? extends Stream<? T> f) :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**retourne la concaténation des streams mapper .apply(x) pour tout x dans this.**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Stream<? T> limit(long n) :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**tranche le stream après n éléments.**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**<? T> map(Function<? T, ? U> f) :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**retourne le stream des éléments f .apply(x) pour tout x élément de this.**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**342. Remarque : c ne sert que pour ses effets de bord. peek peut notamment être utilisé pour le débogage.**

## L'interface Collector

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Calculer et extraire un résultat : les opérations terminales**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Opérations spécialisées (1)**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Operations spéciales (1)**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Optional<? T> reduce(BinaryOperator<? T, ? op) :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**effectue la réduction du stream par l'opération d'accumulation associative op.**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**T reduce(T zero, BinaryOperator<? T, ? op) :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**U reduce(U zero, BiFunction<? U, ? op) :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**BinaryOperator<? U> comb) :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**idem avec accumulation vers autre type.**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**343. Opération apply à l'objet collector ayant retour (p. ex. suppression des doublons).**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**Trois techniques pour fabriquer un tel objet :**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**• (cas courants) utiliser une des fabriques statiques de la classe Collectors.of() ('constructeur' généraliste)**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**• utiliser la fabrique statique Collector.collect() : on parle juste après.**

Compléments en P00

Aidez Degone

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Interfaces et types dérivés Classes et interfaces Concurrence Interfaces graphiques

**343. Opération appelée fold dans d'autres langages. Les définitions varient...**

## La classe Collectors

## Construire un collector via l'interface Collector

Compléments en P00

Ateliers Degorre

Cette classe, non instanciable, est une bibliothèque de fabricries statiques pour obtenir simplement les collectors les plus courants. Quelques exemples : `Collectors.toList()`, `Collectors.toSet()`, `Collectors.counting()`, `Collectors.groupingBy(...)`, `Collectors.reducing(...)`, `Collectors.toConcurrentMap(...)`...

→ on retrouve des opérations équivalentes<sup>344</sup> à la plupart des réductions de l'interface Stream.

Ainsi autre façon d'avoir la taille d'un stream :

```
monStream.collect(Collectors.counting())
```

mais ici : implémentation "mutable" utilisant un attribut accumulateur, alors que dans Stream, les réductions utilisent des fonctions "pures"

`345.` ... mais le plus simple reste `monStream.count()` !

## Effacement de type (type erasure)

### Effacement de type

Compléments en P00

Ateliers Degorre

De quoi il s'agit.

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Héritage

Généricité

Concurrence

Interfacing graphiques

**Effacement d'un type** : sur-approximation permettant d'obtenir un type **réifiable** (i.e. : « classique », façon Java 4) à partir de n'importe quel type. Plus précisément (JLS 4.6) :

- L'effacement d'un type générique ou paramétré de forme `G<...>`, est le type brut `G`.
- L'effacement d'une variable de type est l'effacement de sa borne supérieure.
- L'effacement de tout autre type `T` est `T`.

L'idée principale du phénomène appelé **effacement de type** (ou **type erasure**) c'est que le système de types de la JVM ne connaît que les types réifiables.

Autrement dit : la paramétrisation générique n'a pas d'impact à l'exécution.

Plus de détails juste après.

## Compléments en P00

## Effacement de type (type erasure)

### Effacement de type

Ateliers Degorre

Au cas où la bibliothèque Collector ne contient pas ce qu'on cherche, on peut créer un Collector autrement :

- créer et instancier une classe implémentant Collector.
- Méthodes à implémenter : `accumulator()`, `characteristics()`, `combiner()`, `finished()` et `supplier()`.
- sinon, créer directement l'objet grâce à la méthode statique `Collector.of()` :

```
c2 = Collector<Integer, List<Integer>, Integer>::new, List::add, () -> (l1, l2) -> l1.addAll(l2); return c2;
```

(façon... un peu alambiquée de calculer la taille d'un stream...)

Utiliser la méthode `of()` est plus "léger" syntaxiquement, mais ne permet pas d'ajouter des champs ou des méthodes à l'objet fabriqué.

## Compléments en P00

## Effacement de type (type erasure)

### Effacement de type

Ateliers Degorre

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Héritage

Généricité

Concurrence

Interfacing graphiques

Au cas où la bibliothèque Collector ne contient pas ce qu'on cherche, on peut créer un Collector autrement :

- créer et instancier une classe implémentant Collector.
- Méthodes à implémenter : `accumulator()`, `characteristics()`, `combiner()`, `finished()` et `supplier()`.
- sinon, créer directement l'objet grâce à la méthode statique `Collector.of()` :

```
c2 = Collector<Integer, List<Integer>, Integer>::new, List::add, () -> (l1, l2) -> l1.addAll(l2); return c2;
```

(façon... un peu alambiquée de calculer la taille d'un stream...)

Utiliser la méthode `of()` est plus "léger" syntaxiquement, mais ne permet pas d'ajouter des champs ou des méthodes à l'objet fabriqué.

## Compléments en P00

## Effacement de type (type erasure)

### Effacement de type

Ateliers Degorre

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Héritage

Généricité

Concurrence

Interfacing graphiques

**Remarque :** le code-octet contient tout de même encore des types non réifiables. C'est le cas pour les signatures <sup>346</sup> des classes et de leurs membres. Cela est indispensable pour vérifier les types génériques lors de la compilation des classes dépendantes.

Il est donc faux que le code-octet ne soit déjà plus rien de la paramétrisation générique d'une classe.

Mais cette information est à destination du compilateur, et non pas de la JVM<sup>349</sup>.

348. Signature au sens de la JVM. Pour la JVM, la signature contient toute l'information de type d'une entité donnée. Par exemple, pour une méthode, c'est son type de retour et les types de ses paramètres. Cette notion est donc différente de la notion de signature dans un code source Java. Elle est aussi différente de la notion de descripteur tout juste évoquée.

349. Ceci dit, il est possible de lire les signatures pendant l'exécution (grâce à la réflexion). Mais cela ne permet en aucun cas de savoir quels paramètres de types effectifs ont été utilisés pour instancier un objet dont on exécute une méthode donnée.

## Compléments en P00

## Effacement de type (type erasure)

### Effacement de type

Ateliers Degorre

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Héritage

Généricité

Concurrence

Interfacing graphiques

L'effacement de type commence en réalité dès la compilation.

**Description de méthode** : information, dans la **table des constantes** d'une classe compilée, permettant d'identifier une méthode (peut-être surchargée) de façon unique. Tout appelle de méthode<sup>346</sup> dans le code-octet fait référence à un tel descripteur.

On un descripteur consiste en un couple : (nom de méthode, types réifiables des paramètres).

**Conséquences :**

- aucun paramètre de type n'est réellement passé aux constructeurs (et méthodes)
- les objets ne stockent donc pas les valeurs de leurs paramètres de types. Ils ne peuvent donc connaître que leur classe<sup>347</sup>. Les objets paramétrés » n'existent pas.

346. `invokeSpecial`, `invokeVirtual` et `invokeInterface` prennent un index de la table des constantes comme paramètre.

347. Qui n'existe qu'en seul exemplaire dans la mémoire, quelle que soit la paramétrisation.

## Compléments en P00

## Effacement de type (type erasure)

### Effacement de type

Ateliers Degorre

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Héritage

Généricité

Concurrence

Interfacing graphiques

Dans une classe, on ne peut pas définir plusieurs méthodes dont les signatures seraient identiques après effacement (leurs descripteurs seraient identiques).

```
// ne compile pas
public class A {
    List<String> f1() { return null; }
    List<Integer> f1() { return null; }
}
```

Une classe ne peut pas implementer plusieurs fois une interface générique avec des paramètres différents (on se retrouverait dans le cas précédent).

```
// ne compile pas
public class A extends ArrayList<Integer> implements List<String> { ... }
```

## Compléments en P00

## Effacement de type (type erasure)

### Effacement de type

Ateliers Degorre

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Héritage

Généricité

Concurrence

Interfacing graphiques

Autres conséquences directes de l'effacement dans les descripteurs :

- Dans une classe, on ne peut pas définir plusieurs méthodes dont les signatures seraient identiques après effacement (leurs descripteurs seraient identiques).

```
// ne compile pas
public class A {
    List<Integer> f1() { return null; }
    List<String> f1() { return null; }
}
```

Une classe ne peut pas implementer plusieurs fois une interface générique avec des paramètres différents (on se retrouverait dans le cas précédent).

```
// ne compile pas
public class A extends ArrayList<Integer> implements List<String> { ... }
```

## Compléments en P00

## Effacement de type (type erasure)

### Effacement de type

Ateliers Degorre

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Héritage

Généricité

Concurrence

Interfacing graphiques

Le problème : examinons l'exemple suivant (qui ne compile pas).

```
public static void main(String[] args) {
    List<VoitureBansPermis> listVoitureSP = new ArrayList();
    // Ici : ceci est en fait interdit... mais supposons que ça passe...
    listVoitureSP.add(new Voiture());
    // 12: instruction bien typée (pour le compilateur), mais...
    // 13: ... logiquement ça afficherait "voiture" (contradictoire)
    System.out.println(listVoitureSP.get(0).getClasse());
}
```

S'il compilait, en l'exécutant, à la fin, `listVoitureSP = listVoiture` contenaitrait des voitures → **contredit la déclaration de listVoiture**!

Ainsi, Java interdit l1 : deux spécialisations différentes du même type générique sont incompatibles. On dit que les génériques de Java sont **invariants**.

## Tableaux – génériques ou pas ? (2)

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | <p>Tableaux – génériques ou pas ? (1)</p> <p><b>Remarque :</b> l'analogue à l'exemple précédent utilisant <code>Voiture[]</code> au lieu de <code>List&lt;Voiture&gt;</code> compile sans avertissement :</p> <pre>public static void main(String[] args) {     VoitureSansPermis[] listVoituresSP = new VoitureSansPermis[100];     // L1: ceri est autorisé !     Voiture[] listVoiture = listVoituresSP;     // I2: instruction bien typée (pour le compilateur), mais... ArrayStoreException !     listVoiture[0] = new Voiture();     // I3: on ne va pas jusque là     System.out.println(listVoituresSP[0].getClass()); }</pre> <p><b>Note :</b> cependant le compilateur détecte la conversion « louche » et signale un avertissement (warning) « <b>unchecked conversion</b> » pour la ligne I1.</p> <p><b>Moralité :</b> si avertissement, alors garanties usuelles supprimées.</p> <p><b>ClassCastException</b> peut se produire à l'exécution.</p> <p>→ Genre de » générité, mais conception obsolète : avec la générité moderne, la compilation garantit une exécution sans erreur.</p> <p>351. Raison : un tableau est à la fois producteur et consommateur. D'un point de vue théorique, une telle structure de données ne peut être qu'invariante, si on veut des garanties dès la compilation.</p> |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | <p>Tableaux – génériques ou pas ? (2)</p> <p><b>Génériques et tableaux</b></p> <p><b>Remarque :</b> l'analogue à l'exécution, mais le + iot possible)</p> <ul style="list-style-type: none"> <li>usage normal : conversion sans warning de <code>SousType[]</code> à <code>SuperType[]</code> par upcasting (implicite), Possibilité d'<code>ArrayStoreException</code> à l'exécution.</li> </ul> <pre>Object[] tab = new String[10]; tab[0] = Integer.valueOf(3); // BOOM ! (ArrayStoreException)</pre> <p>Pas idéal, mais aurait pu être pire : le crash évité que le programme continue avec une mémoire incomplète.</p> <p>usage anominal : avec cast explicite vers type incompatible : <code>(String[])(Object) new Integer[10]</code> compile mais avec warning et fait <code>ClassCastException</code> quand on exécute (tout va bien : on avait été prévenu).</p> |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | <p>Types génériques et sous-type</p> <p>Invariance des génériques</p> <p><b>type erasure → vérification à la compilation seulement. Court-circuitons-la, pour voir :</b></p> <pre>// I1 : version avec "triche" (pas d'exception car type érasure) List&lt;Voiture&gt; listVoiture = (List&lt;Voiture&gt;)listVoitureSP; // I2: * / listVoiture.add(new Voiture()); // I3: ça affiche effectivement "voiture" (oooh !) System.out.println(listVoitureSP.get(0).getClass()); // I4: et pour la forme, une petite ClassCastException : VoitureSansPermis vsp = listVoitureSP.get(0); // I5: on ne va pas jusque là System.out.println(listVoituresSP[0].getClass());</pre> <p><b>Supposons <code>T extends Up</code> paramètre de type.</b></p> <p><b><code>new T[10]</code> est aussi interdit.</b></p> <p><b>Raison :</b> après compilation, <code>T</code> est oublié et remplacé par <code>Up</code>. Au mieux <code>new T[10]</code> pourrait être compilé comme <code>new Up[10]</code>. Mais si c'était ce qui se passait, on pourrait trop facilement « polluer » la mémoire sans s'en rendre compte :</p> <pre>static {&gt;&gt; T[] makeArray(int size) { return new T[10]; /* interdit ! */         static {&gt;&gt; TString tString = makeArray(10); // à l'exécution on affectera un Object[]         Object[] object = tString; // toujours autorisé covariance)         object[0] = 42; // et BOOM ! Maintenant tString contient un Integer !     }/* ... et là, c'est le drame !     (ClassCastException)     String s = tab[0].new Up[10];     String s = bsa[0].x;</pre> <p>En pratique, pour faire compiler cela, il faut « tricher » avec <code>cast</code> explicite : <code>T[] tab = (T[])new Up[10];</code>.</p> <p>Ça l'hérisse pas le problème ci-dessus, mais au moins le compilateur affiche un warning (<b>unchecked conversion</b>).</p> |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | <p>Tableaux – génériques ou pas ? (2)</p> <p><b>Génériques et tableaux</b></p> <p><b>Remarque :</b> l'analogue à l'exécution, mais le + iot possible</p> <p><b>Covariance à la place d'invariance : ⇒ vérifications moins strictes à la compilation, rendant possibles des problèmes à l'exécution</b></p> <ul style="list-style-type: none"> <li>pour détecter les problèmes au plus tôt : à l'instanciation, un tableau enregistre le nom du type déclaré pour ses éléments (pas d'effacement de type)</li> <li>cela permet à la JVM de déclencher <code>ArrayTypeException</code> lors de toute tentative d'y stocker un élément du mauvais type, au lieu de <code>ClassCastException</code> lors de son utilisation (donc bien plus tard).</li> </ul> <p>→ « Genre de » générité, mais conception obsolète : avec la générité moderne, la compilation garantit une exécution sans erreur.</p> <p>351. Raison : un tableau est à la fois producteur et consommateur. D'un point de vue théorique, une telle structure de données ne peut être qu'invariante, si on veut des garanties dès la compilation.</p> |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | <p>Tableaux – génériques ou pas ? (2)</p> <p><b>Génériques et tableaux</b></p> <p><b>Remarque :</b> l'analogue à l'exécution, mais le + iot possible</p> <p><b>Covariance à la place d'invariance : ⇒ vérifications moins strictes à la compilation, rendant possibles des problèmes à l'exécution</b></p> <ul style="list-style-type: none"> <li>pour détecter les problèmes au plus tôt : à l'instanciation, un tableau enregistre le nom du type déclaré pour ses éléments (pas d'effacement de type)</li> <li>cela permet à la JVM de déclencher <code>ArrayTypeException</code> lors de toute tentative d'y stocker un élément du mauvais type, au lieu de <code>ClassCastException</code> lors de son utilisation (donc bien plus tard).</li> </ul> <p>→ « Genre de » générité, mais conception obsolète : avec la générité moderne, la compilation garantit une exécution sans erreur.</p> <p>351. Raison : un tableau est à la fois producteur et consommateur. D'un point de vue théorique, une telle structure de données ne peut être qu'invariante, si on veut des garanties dès la compilation.</p> |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en P00 | <p>Tableaux – génériques ou pas ? (2)</p> <p><b>Génériques et tableaux</b></p> <p><b>Remarque :</b> l'analogue à l'exécution, mais le + iot possible</p> <p><b>Covariance à la place d'invariance : ⇒ vérifications moins strictes à la compilation, rendant possibles des problèmes à l'exécution</b></p> <ul style="list-style-type: none"> <li>pour détecter les problèmes au plus tôt : à l'instanciation, un tableau enregistre le nom du type déclaré pour ses éléments (pas d'effacement de type)</li> <li>cela permet à la JVM de déclencher <code>ArrayTypeException</code> lors de toute tentative d'y stocker un élément du mauvais type, au lieu de <code>ClassCastException</code> lors de son utilisation (donc bien plus tard).</li> </ul> <p>→ « Genre de » générité, mais conception obsolète : avec la générité moderne, la compilation garantit une exécution sans erreur.</p> <p>351. Raison : un tableau est à la fois producteur et consommateur. D'un point de vue théorique, une telle structure de données ne peut être qu'invariante, si on veut des garanties dès la compilation.</p> |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Les génériques invariants c'est bien mais...

Compléments en P00  
Aldric Degorre

Invariance des génériques → garanties fortes : très bien, mais... très rigide à l'usage !

**Le besoin :** quand  $B <: A^{353}$ , on aimerait pouvoir écrire

```
Gen<T> g = new Gen<B>();
```

- Cela favoriserait le polymorphisme (par sous-type).
- On le fait bien avec les tableaux (`Object[] t = new String[10];`).
- C'est souvent conforme à l'intuition (cf. tableaux).

**Mais on sait que ça risque d'être difficile :**

- On a vu un contre-exemple pathologique (on provoque facilement `ClassCastException` si on force le compilateur à outrepasser l'invariance).
- On a vu les problèmes que posent les tableaux covariants (`(ArrayList<StoreException> possible même dans programme sans warning).`
- Ou bien, peut-être parfois, quand  $B <: A$ .

## Considérations autour de la variance (1)

| Compléments en P00                                                  | Aldric Degorre                                                      | Considérations autour de la variance (1)                                                                                                                          |
|---------------------------------------------------------------------|---------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Introduction                                                        | Introduction                                                        | Pour les quelques pages qui suivent, <b>oublions que javac impose l'invariance.</b>                                                                               |
| Généralités Style Objets et classes Types et polymorphisme Héritage | Généralités Style Objets et classes Types et polymorphisme Héritage | <b>Question :</b> parmi les variables $x$ , $y$ , $z$ et $t$ , ci-dessous, lesquelles devrait-on, idéalement <sup>354</sup> , pouvoir affecter à quelles autres ? |

```
class A {}  
class B extends A {}  
// Interface pour fonctions F -> U (extraite de java.util.function) :  
interface Function<T,U> { U apply(T t); }  
class Test {  
    Function<A,A> x;  
    Function<B,B> y;  
    Function<A,B> z;  
    Function<B,A> t;  
}
```

**Le critère :** on cherche les cas où une instance `Function<Y,Y>` fournit au moins le service d'une instance de `Function<T,T>`.

354. par exemple dans un langage où les génériques pourraient ne pas être invariants

## Considérations autour de la variance (2)

| Compléments en P00 | Aldric Degorre | Considérations autour de la variance (2)                                                                                                   |
|--------------------|----------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| Introduction       | Introduction   | Réponse : affecter $u$ à $v$ a un sens si la méthode <code>apply</code> de $U$ peut remplacer celle de $V$ (en toute situation). C.-à-d. : |

- si elle accepte tous les paramètres effectifs acceptés par celle-ci
- et si les valeurs rentrées appartiennent à un type au moins aussi restreint.

(En résumé : une instance de `Function<Y,Y>` peut remplacer une instance de `Function<Z,T>` si  $X >: Z$  et  $Y <: T$ .)

→ en appliquant ce principe, on voudrait donc que le compilateur accepte :

```
Z = t; z = x; t = y; x = y; z = y;
```

Attention, ces concepts ne sont que théoriques.

## Considérations autour de la variance (3)

| Compléments en P00                                                               | Aldric Degorre                                                                   | Considérations autour de la variance (3)                            |
|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|---------------------------------------------------------------------|
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | <b>Représentation graphique :</b> type générique → pièce de puzzle. |

- Paramètre utilisé en entrée (= type de paramètre de méthode ou type d'attribut public modifiable) → encoché.
- Paramètre utilisé en sortie (= type de retour de méthode, type d'attribut public quelconque) → excroissance.

L'encoché (resp. excroissance) pour un type donné doit contenir celles de ses sous-types.

**Exemple :**



(L'encoché à gauche représente  $T$  et l'excroissance à droite,  $U$ .)

355. Attention, on ne parle pas de Java, mais seulement d'un système de type « idéal ».

## Considérations autour de la variance (4)

| Compléments en P00                                                               | Aldric Degorre                                                                   | Considérations autour de la variance (4)                                                              |
|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------|
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | Ainsi, inclusion des formes si et seulement si il y a sous-typepage :<br>type de expr1  type de expr2 |

varX = expr2;?  
type de varX (type attendu, en négatif) varX = expr1;?  
type de varX = expr2; doit fonctionner 355 (pas de chevauchement) :  
→ seul varX = expr2; doit fonctionner 355 (pas de chevauchement).

## Considérations autour de la variance (5)

| Compléments en P00                                                               | Aldric Degorre                                                                   | Considérations autour de la variance (5)                                      |
|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|-------------------------------------------------------------------------------|
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | La variance souhaitée n'est donc pas la même pour tous les types génériques : |

- intuitif et logique de voir/voir
- `Function<Object, Integer> <: Function<double, Number>`.

**Justification :** le premier paramètre de type est utilisé uniquement pour l'argument de `apply` alors que l'autre est uniquement son type de retour.  
→ Emboîtement de `Function<T, U>` dans `Function<V,W>` possible dès que  $T >: V$  et  $U <: W$ .

Remarque : tailles de l'encoche et de l'excroissance de `Function<T, U>` indépendantes l'une de l'autre car elles représentent 2 paramètres différents. Si le même paramètre de type est utilisé en entrée et en sortie, ça ne marche plus (cf. page d'après).

## Encore un peu de vocabulaire autour de la variance (1)

| Compléments en P00                                                               | Aldric Degorre                                                                   | Encore un peu de vocabulaire autour de la variance (1) |
|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|--------------------------------------------------------|
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | → 3 catégories de paramètres de type :                 |

- paramètre covariant** (comme  $U$ ) : utilisé seulement en position covariante → plus le paramètre effectif est petit, plus le type paramètre devrait être petit;
- paramètre contravariant** (comme  $T$ ) : utilisé seulement en position contravariante → plus le paramètre effectif est petit, plus le type paramètre devrait être grand;
- paramètre invariant** : utilisé à la fois en position covariante et contravariante.

Attention, ces concepts ne sont que théoriques.

Il se trouve que ceux-ci n'ont pas de sens pour le compilateur de Java : rappellez-vous qu'on avait dit que, pour l'instant, on oubliait l'invariance imposée par Java.

## Considérations autour de la variance (6)

| Compléments en P00                                                               | Aldric Degorre                                                                   | Considérations autour de la variance (6)                                                                                                                            |
|----------------------------------------------------------------------------------|----------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | Introduction Généralités Style Objets et classes Types et polymorphisme Héritage | Dans <code>Function&lt;T, U&gt;</code> , $T$ et $U$ ont des influences contraires l'une de l'autre à cause de leur usage dans la méthode de <code>Function</code> . |

- position covariante** : utilisé comme type de retour de méthode (ou type d'attribut) → 2 catégories d'usage :
  - en **position covariante** : utilisé comme type d'un argument dans la signature d'une méthode (ou comme type d'un attribut non `final`)
  - en **position contravariante** : utilise comme type de retour de méthode (ou type d'attribut) → impossible d'encastre la pièce de `List<X>` dans le trou `List<Y>` dans le trou `List<X>` dans le trou `List<Y>` →  $SI X \neq Y$ .

356. Même topo pour `Integer[] <: Number[]` avec les opérations  $x = t[1]$  et  $t[1] = x$ ... mais ça c'est autorisé : en contrepartie, il est nécessaire de faire des vérifications à l'exécution, avec risque de `ArrayStoreException`.

## Gérer la variance dans un programme

### Compléments en P00

Aidez Degorre

### Wildcards

#### principe de base (1)

#### Wildcards

principe de base (2)

- 2 approches principales pour prendre en compte le phénomène de la variance :

- annotations de variance sur le site de déclaration** (n'existent pas en Java)

Variance définie (définitivement) dans la déclaration du type générique.

Exemple en langage Kotlin, on utilise **in** (contravariance) et **out** (covariance) :

```
interface Function<in T, out U> { fun apply(t: T) : U }
```

Alors, dans cet exemple, `Function<A, B> <: Function<B, A>`.<sup>357</sup>

- annotations de variance sur le site d'utilisation**

Variance choisie lors de l'usage d'un type générique (dans déclarations de variables et signatures de méthodes).

C'est l'approche utilisée par Java, via le mécanisme des **wildcards**.

<sup>357</sup> Le compilateur de Kotlin vérifie que les paramètres covariants (resp. contravariants) sont effectivement uniquement utilisés en position covariante (resp. contravariante).

### Compléments en P00

Aidez Degorre

### Wildcards

#### principe de base (1)

#### Wildcards

principe de base (2)

### (On revient enfin à Java !)

- Quand on écrit un type paramétré, les paramètres peuvent en fait être soit des types, soit le symbole « ? » (symbolisant un joker, un **wildcard**) , parfois munis d'une **borne**.

- Les types paramétrés dont le paramètre est compatible avec la borne du **wildcard** se comportent alors comme des sous-types du type contenant le **wildcard**.  
Ainsi `List<Integer>` est sous-type de `List<?>`.

**Remarque :** « ? » tout seul n'est pas un type. Ce caractère ne peut être utilisé que pour écrire un type paramétré (entre « < » et « > »).

ou même

ou bien

ou encore

ou alors

ou toujours

ou toujours

ou encore

ou alors

### Toute occurrence de « ? » peut se voir associer une borne.

Le principe est similaire aux bornes de paramètres de type, avec quelques différences :

- « ? » **bornable** à chaque usage (or, paramètres bornables juste à leur introduction).
- Les « ? » admettent des bornes supérieures (`T<? extends A>`), mais aussi des bornes inférieures (`T<? super A>`), imposant que toute concréétisation doit être un supertype de la borne.

Pour un « ? » Java autorise une seule borne à la fois.<sup>358</sup>

S'il y a plusieurs types concrets comme bonnes supérieures, il est possible de contourner cette limite en introduisant un type intermédiaire : **interface Borne1, Borne2** …

Combiner plusieurs bonnes inférieures concrètes (disons `a` et `b`) ne sera t-à-dire : `a: Borne1, Borne2, ...`

c'est le plus petit type contenant `A|B` (qui n'est pas un type de Java). Ainsi `<? super C>` serait équivalent à `<? super A|B>` (syntaxe tictac).

Simplement, pour mixer des bonnes qui sont elles mêmes des paramètres, d'autres techniques basées sur l'introduction d'une variable de type supplémentaire sont envisagables.

Maintenant, les affectations qu'on voulait écrire sont acceptées par le compilateur :

```
interface Function<T, B> { U apply(T t); }
class Test { Function<A, X> f; Function<A, B> g; Function<B, Y> h; }

Test f = new Test();
f.f.apply(z);
f.g.apply(z);
f.h.apply(z);
```

En réalité, pour une expression, avoir le type `Gen<?` veut dire qu'il **existe**<sup>360</sup> un type

`QueLqueChose` (inconnu mais fixé) tel que cette expression est de type `Gen<QueLqueChose>`.

Il faut interpréter le type d'une expression à **wildcards** comme un type inconnu appartenant à l'ensemble des types respectant les contraintes trouvées.

### Wildcards

#### compléments en P00

Aidez Degorre

### Wildcards

#### Conversion par capture (1)

#### Wildcards

Conversion par capture (2)

### Wildcards

#### utilisation pour signaler la variance

Aidez Degorre

### Wildcards

#### conversion par capture

#### Wildcards

conversion par capture

Pour revenir au problème initial, reprenons notre exemple :

```
interface Function<T, B> { U apply(T t); }
class Test { Function<A, X> f; Function<A, B> g; Function<B, Y> h; }

Test f = new Test();
f.f.apply(z);
f.g.apply(z);
f.h.apply(z);
```

Maintenant, les affectations qu'on voulait écrire sont acceptées par le compilateur :

```
z = 1; z = y; t = z; // vérifiez !
t = a; a = x; apply(x); a = z; apply(y); b = t; apply(b);
```

Recette : position covariante → **extends**, position contravariante → **super**.

Se rappeler **PECs** : « produire extends, consumer super ».

Inversez **super** et **extends** et vérifiez que les appels à **apply** ne fonctionnent plus.

### Compléments en P00

Aidez Degorre

### Wildcards

#### généralités

#### Wildcards

généralités

### Wildcards

#### styling

#### Wildcards

styling

### Wildcards

#### objets et classes

#### Wildcards

objets et classes

### Wildcards

#### types et polymorphisme

#### Wildcards

types et polymorphisme

### Wildcards

#### héritage

#### Wildcards

héritage

### Wildcards

#### généralité

#### Wildcards

généralité

### Wildcards

#### concurrente

#### Wildcards

concurrente

### Wildcards

#### interfaces graphiques

#### Wildcards

interfaces graphiques

### L'affectation suivante estelle bien typée ?

Pour savoir, on vérifie si le terme droit de l'affectation à un type compatible avec son emplacement.

Son type est `ArrayList<String>`, or le type attendu dans le contexte soit de la forme `TA?` ou `List<? extends Serializable>` (= type de la variable à affecter).

- D'une part, `String` satisfait la borne de `? (String implémente Serializable)` et, d'autre part, `ArrayList<String> <: List<String>`.

Donc cette affectation est bien typée.

L'affectation suivante est elle bien typée ?

Pour savoir, on vérifie si le terme droit de l'affectation à un type compatible avec son emplacement.

Son type est `ArrayList<String>`, or le type attendu dans le contexte soit de la forme `TA?` ou `List<? extends Serializable>`.

- Combien plusieurs bonnes inférieures concrètes (disons `a` et `b`) ne sera t-à-dire : `a: Borne1, Borne2, ...`

C'est le plus petit type contenant `A|B` (qui n'est pas un type de Java). Ainsi `<? super C>` serait équivalent à `<? super A|B>` (syntaxe tictac).

Simplement, pour mixer des bonnes qui sont elles mêmes des paramètres, d'autres techniques basées sur l'introduction d'une variable de type supplémentaire sont envisagables.

### Pour revenir au problème initial, reprenons notre exemple :

```
interface Function<T, B> { U apply(T t); }
class Test { Function<A, X> f; Function<A, B> g; Function<B, Y> h; }

Test f = new Test();
f.f.apply(z);
f.g.apply(z);
f.h.apply(z);
```

### En réalité, pour une expression, avoir le type `Gen<?` veut dire qu'il **existe**<sup>360</sup> un type

`QueLqueChose` (inconnu mais fixé) tel que cette expression est de type `Gen<QueLqueChose>`.

Il faut interpréter le type d'une expression à **wildcards** comme un type inconnu appartenant à l'ensemble des types respectant les contraintes trouvées.

359. Explication un peu plus loin. Ceci concerne le cas où le type de l'expression contient un ?.

360. Pour les logiciens, cette transformation s'apparente à la skolemisation : on remplace une variable quantifiée existentiellement par un nouveau symbole.

362. On ne regarde pas en profondeur : `List<?>` devient `List<capture#1-of ?>`. La compilation se rapelle que `capture#1-of ?` est sous-type de `Set<?`.

363. Cela se produit quand l'expression est une variable type avec un nom, un appel de méthode dont le type de retour contient des ?, ou une expression casée vers un telle type.

361. Pour les logiciens, cette transformation s'apparente à la skolemisation : on remplace une variable quantifiée existentiellement par un nouveau symbole.

362. On ne regarde pas en profondeur : `List<?>` devient `List<capture#1-of ?>`. La compilation se rapelle que `capture#1-of ?` est sous-type de `Set<?`.

363. Cela se produit quand l'expression est une variable type avec un nom, un appel de méthode dont le type de retour contient des ?, ou une expression casée vers un telle type.

### Wildcards

#### styling

#### Wildcards

styling

### Wildcards

#### objets et classes

#### Wildcards

objets et classes

### Wildcards

#### types et polymorphisme

#### Wildcards

types et polymorphisme

### Wildcards

#### héritage

#### Wildcards

héritage

### Wildcards

#### généralité

#### Wildcards

généralité

### Wildcards

#### concurrente

#### Wildcards

concurrente

### Wildcards

#### interfaces graphiques

#### Wildcards

interfaces graphiques

## Wildcards

Quid des exemples plus complexes ? (1)

## Wildcards

Exemple pathologique

## Wildcards

Conversion par capture (3)

## Wildcards

Compléments en P00 Affiche Degré de

## Wildcards

Compléments en P00 Affiche Degré de

### Exemple :

- soit une expression :  
**new HashMap<? super String, ? extends List<?>, ? extends List<?>?>**,
- son type « brut » : **HashMap<? super String, ? extends List<?>, ? extends List<?>?>**,
- son type après conversion par capture :  
**HashMap<capture#1-0?, capture#2-0?, capture#2-0?, capture#1-0? > String et capture#2-0? > String et capture#2-0? <: List<?>**.
- Le compilateur se rappelle que **capture#1-0? > String et capture#2-0? <: List<?>**.

Cette conversion a lieu à chaque fois que le type d'une expression est évalué. Ainsi, une expression composite peut contenir plusieurs captures différentes accumulées depuis l'analyse du type de ses sous-expressions.

```
List<? super String> l = new ArrayList<?>(); l.add("toto"); // ok
```

### Explication à la 2e ligne,

- la 1e occurrence de **l** est de type **List<capture#1-0?>**  $\Rightarrow$  **l.add(...)** attend un paramètre de type **capture#1-0?** ;
- la 2e occurrence de **l** est de type **List<capture#2-0?>** (capture indépendante)  $\Rightarrow$  **l.get(0)** est de type **capture#2-0?** ;
- or **capture#1-0?** et **capture#2-0?** sont, du point de vue du compilateur, deux types quelconques sans lien de parenté, d'où l'erreur de type.

On peut contourner en forçant une capture anticipée (via méthode auxiliaire) :

```
// méthode auxiliaire. Ici, tout est ok, car L.get(0) de type T, or l.add() prend du T.  
<? super void aux(List<?> l) { l.add(l.get(0)); }  
plus loin  
List<? super String> l = new ArrayList<?>(); l.add("toto"); aux(l); // encore ok
```

## Wildcards

Quid des exemples plus complexes (1)

## Wildcards

Compléments en P00 Affiche Degré de

## Wildcards

Compléments en P00 Affiche Degré de

### Exemple :

- soit une expression :  
**new HashMap<? super String, ? extends List<?>, ?>**,
- son type « brut » : **HashMap<? super String, ? extends List<?>, ?>**,
- son type après conversion par capture :  
**HashMap<capture#1-0?, capture#2-0?, capture#2-0?, capture#1-0? > String et capture#2-0? > String et capture#2-0? <: List<?>**.
- Le compilateur se rappelle que **capture#1-0? > String et capture#2-0? <: List<?>**.

Cette conversion a lieu à chaque fois que le type d'une expression est évalué. Ainsi, une expression composite peut contenir plusieurs captures différentes accumulées depuis l'analyse du type de ses sous-expressions.

```
List<? super String> l = new ArrayList<?>(); l.add("toto"); // ok
```

### Explication à la 2e ligne,

- soit une expression :  
**new HashMap<? super String, ? extends List<?>, ? extends List<?>?>**,
- son type « brut » : **HashMap<? super String, ? extends List<?>, ? extends List<?>?>** ; Mais typé ! Mais pour quoi ?
- son type après conversion par capture :  
**HashMap<capture#1-0?, capture#2-0?, capture#2-0?, capture#1-0? > String et capture#2-0? > String et capture#2-0? <: List<?>**.
- Le compilateur se rappelle que **capture#1-0? > String et capture#2-0? > String et capture#2-0? <: List<?>**.

On peut contourner en forçant une capture anticipée (via méthode auxiliaire) :

```
// méthode auxiliaire. Ici, tout est ok, car L.get(0) de type T, or l.add() prend du T.  
<? super void aux(List<?> l) { l.add(l.get(0)); }  
plus loin  
List<? super String> l = new ArrayList<?>(); l.add("toto"); aux(l); // encore ok
```

## Concurrence

Où et quand ?

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Quid des exemples variées en programmation 366 :

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Compléments en P00 Affiche Degré de

### Definition (Concurrence)

- Deux actions, instructions, travaux, tâches, processus, etc. sont **concurrents** si leurs executions sont **indépendantes** l'une de l'autre (un n'attend pas de résultat de l'autre).

Dans l'exemple (2 ligne, à droite de =) :

```
List<? extends A>? super String> las = new ArrayList<?>();  
List<? extends A>? super Integer> lar = las;
```

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Quid des exemples plus complexes (2)?

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Compléments en P00 Affiche Degré de

### Consequence :

- Deux actions concurrentes peuvent s'exécuter simultanément, si la plateforme d'exécution le permet.
- Un programme concurrent est un programme dont certaines portions de code sont indépendantes l'une des autres et tel que la plateforme d'exécution sait exploiter ce fait pour optimiser l'exécution. 364

364. Le plus souvent, cette connaissance nécessite que les portions concurrentes soient signalées dans le code source.

365. Notamment en exécutant simultanément, en parallèle, ces portions de code si c'est possible.

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Quid des exemples plus complexes (2)?

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Compléments en P00 Affiche Degré de

### Degre de parallelisme

- Simultanéité au niveau le plus bas : si 2 travaux s'exécutent en parallèle, à un instant t, s'exécutent en même temps une instruction de l'un et de l'autre.
- exécution sur 2 lieux physiques différents (e.g. 2 coeurs, 2 circuits, ...).

Pour des raisons économiques et technologiques, les microprocesseurs modernes (multi-cœur 370) ont typiquement un degré de parallélisme  $\geq 2$ .

C'est une opportunité qui faut savoir saisir !

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Quid des exemples variées en programmation 366 :

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Compléments en P00 Affiche Degré de

### Nécessité :

- pouvoir programmer des fonctionnalités intrinsèquement concurrentes (serveur web, IG, etc.).

Ainsi, l'enjeu de la programmation concurrente est double :

En effet : des travaux indépendants (concurrents) peuvent naturellement être confiés à des unités d'exécution distinctes (parallèles),

Malheureusement, la programmation concurrente est un art difficile... .

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Quid des exemples variées en programmation 366 :

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Compléments en P00 Affiche Degré de

### Opportunisme :

- tirer partie de toute la puissance de calcul du matériel contemporain.

369. Pour des raisons économiques et technologiques, les microprocesseurs modernes (multi-cœur 370) ont typiquement un degré de parallélisme  $\geq 2$ .

C'est une opportunité qui faut savoir saisir !

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Quid des exemples variées en programmation 366 :

## Concurrence

Compléments en P00 Affiche Degré de

## Concurrence

Compléments en P00 Affiche Degré de

### Particularité :

- Exécuter deux travaux réellement concurrents en parallèle est facile 371 , mais la réalité est souvent plus compliquée :

371. On en affecte un à chaque cœur, pourvu qu'il y ait 2 coeurs disponibles, et on n'en parle plus !

372. Sinon ce ne sont des sous-programmes mais des programmes indépendants à part entière !

373. En fait, ces deux aspects sont indissociables.

|                                                                                                                                                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h3>Questions d'ordonnancement</h3> <p>Compléments en P00<br/>Multi-tâche préemptif vs. coopératif</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Géométrie Concurrence et synchronization les threads Théorie en Java Théorie en C/C++ avec les threads Interfaces graphiques Gestion des erreurs et exceptions</p>              | <p><b>Un ordonnanceur</b> est un programme chargé de répartir les tâches concurrentes sur les unités d'exécutions disponibles. Il s'agit souvent d'un sous-système du noyau de l'OS.<sup>374</sup></p> <p>L'ordonnanceur peut mettre en œuvre :</p> <ul style="list-style-type: none"> <li>• un fonctionnement <b>multi-tâches préemptif</b> : l'ordonnanceur choisit quand mettre en pause une tâche pour reprendre l'exécution d'une autre. Cela peut arriver (presque) à tout moment.</li> <li>• C'est le cas pour la gestion des processus dans les OS modernes pour ordinateur personnel.</li> <li>• ou bien un fonctionnement <b>multi-tâches coopératif</b> : chaque tâche signale à l'ordonnanceur quand elle peut être mise en attente (par exemple en faisant un appel bloquant).</li> </ul> <p><sup>374.</sup> Operating System/système d'exploitation</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <h3>Questions de synchronisation</h3> <p>Compléments en P00<br/>Transmettre des résultats d'une tâche à l'autre</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Géométrie Concurrence et synchronization les threads Théorie en Java Théorie en C/C++ avec les threads Interfaces graphiques Gestion des erreurs et exceptions</p> | <p>Plusieurs techniques de transmission de résultats :</p> <ul style="list-style-type: none"> <li>• <b>variables partagées</b> : variables accessibles par plusieurs tâches concurrentes. Données partagées de façon transparente, sans synchronisation a priori, mais le langage permet d'insérer des primitives de synchronisation explicite.<sup>375</sup></li> <li>• <b>passage de message</b> : données « envoyées »<sup>376</sup> d'une tâche à l'autre. Synchronisation implicite de l'émission et de la réception du message : par exemple, une tâche en attente de réception est bloquée tant qu'elle n'a rien reçu.<sup>377</sup></li> </ul> <p>La réalité physique est plus proche du modèle des variables partagées<sup>378</sup>, mais le passage de message est un paradigme plus sûr.<sup>379</sup></p> <p>375. En Java : <code>start()</code>, <code>join()</code>, <b>volatile synchronize</b> et <code>wait()</code>/<code>notify()</code>.</p> <p>376. Sous-entendu : l'enoyer ne peut plus accéder à ce qui a été envoyé.</p> <p>377. C'est une possibilité. On peut aussi bloquer la tâche émettrice (canal borné, « rendez-vous »).</p> <p>378. Mémoire central visible par plusieurs CPU, la tâche émettrice</p> <p>379. Pour lequel la sûreté d'un programme est plus facile à prouver.</p>                                                                                                                                                                                                                                                                  |
| <h3>Questions de synchronisation</h3> <p>Compléments en P00<br/>Envoyer des messages</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Géométrie Concurrence et synchronization les threads Théorie en Java Théorie en C/C++ avec les threads Interfaces graphiques Gestion des erreurs et exceptions</p>                            | <p><b>fonctions bloquantes</b> : la tâche réceptrice appelle une fonction fournie par la bibliothèque, qui la bloque jusqu'à ce que la valeur attendue soit disponible.</p> <pre>ForJoinTask&lt;Result&gt; task = ForkJoinTask.adapt(() -&gt; {     return () -&gt; {         Result result;         ... // plus loin : join() :// appeler à fonction bloquante join()         ... // tache 2 : fait avec le résultat de tache 1     }; });</pre> <p>• <b>fonctions de rappel (callbacks)</b> : on passe à la bibliothèque une fonction que celle-ci appellera sur le résultat attendu dès qu'il sera disponible.</p> <pre>CompletableFuture.supplyAsync(() -&gt; {     ... // tache 1     return () -&gt; {         ... // corps de la fonction de rappel         ... // tache 2 : faire qc avec le resultat de tache 1     }; });</pre> <p><b>Exemple de deux threads, l'un qui compte jusqu'à 10 alors que l'autre récite l'alphabet :</b></p> <pre>class ReciteAlphabet extends Thread {     @Override     public void run() {         for (int i = 0; i &lt; 10; i++) {             System.out.print(i + " ");         }     } }  class Example {     public static void main(String[] args) {         new ReciteAlphabet().start();         new ReciteAlphabet().start();     } }</pre> <p>mais également</p> <pre>public void run() {     for (int i = 0; i &lt; 26; i++) {         System.out.print((char)(i + 'a'));     } }</pre> <p>ou encore</p> <pre>public void run() {     for (int i = 0; i &lt; 26; i++) {         System.out.print((char)(i + 'a'));     } }</pre> |
| <h3>Questions de synchronisation</h3> <p>Compléments en P00<br/>Aider Degorre</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Géométrie Concurrence et synchronization les threads Théorie en Java Théorie en C/C++ avec les threads Interfaces graphiques Gestion des erreurs et exceptions</p>                                   | <p><b>Mais on peut simuler le passage de message :</b></p> <pre>public final class MailBox&lt;T&gt; implements Serializable {     private T content; // classe réutilisable, simulant un passage de message avec *rendez-vous*     public synchronized void sendTossage(T message) throws InterruptedException {         while (content == null) { // attend la condition content == null             notify(); // débloque les autres threads en attente sur cette MailBox         }         content = message; // débloque les autres threads en attente sur cette MailBox     }     public synchronized T receiveMessage() throws InterruptedException {         while (content == null) { // attend la condition content == null             wait(); // débloque les autres threads en attente sur cette MailBox         }         return content;     } }</pre> <p><b>Notion de thread</b></p> <p>Compléments en P00<br/>Autre-Degorre</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Géométrie Concurrence et synchronization les threads Théorie en Java Théorie en C/C++ avec les threads Interfaces graphiques Gestion des erreurs et exceptions</p>                                                                                                                                                                                                                                                                                                                                                                               |

|                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h3>Notion de thread</h3> <p>Compléments en P00<br/>Autre-Degorre</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Géométrie Concurrence et synchronization les threads Théorie en Java Théorie en C/C++ avec les threads Interfaces graphiques Gestion des erreurs et exceptions</p> | <p><b>Definition [Thread] ou fil d'exécution</b></p> <p>Abstraction concurrente consistant en une séquence d'instructions dont l'exécution simule une exécution séquentielle (en interne).<sup>381</sup> et parallèle à celle des autres threads.</p> <ul style="list-style-type: none"> <li>• Un nombre quelconque de threads s'exécute sur une plateforme de degré de parallélisme quelconque.<sup>382</sup> Un ordonnanceur partage les ressources de la plateforme pour que cela soit possible.</li> <li>• Ainsi, n threads en exécution simultanée simulent un parallélisme de degré n</li> <li>• Un processus<sup>383</sup> (= 1 application en exécution) peut utiliser plusieurs threads qui ont accès aux mêmes données (mémoire partagée).</li> </ul> <p><sup>381.</sup> Ce qui permet de le programmer avec les principes habituels de programmation impérative : séquences d'instructions, boucles, branchements, pile d'appels de fonctions, ...</p> <p><sup>382.</sup> Même inférieur au nombre de threads</p> <p><sup>383.</sup> Cette fois-ci au sens où on tient en mémoire.</p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h3>Notion de Thread</h3> <p>Compléments en P00<br/>Exemple simple (2)</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Géométrie Concurrence et synchronization les threads Théorie en Java Théorie en C/C++ avec les threads Interfaces graphiques Gestion des erreurs et exceptions</p> | <p><b>Notion de thread</b></p> <p>Compléments en P00<br/>Autre-Degorre</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Géométrie Concurrence et synchronization les threads Théorie en Java Théorie en C/C++ avec les threads Interfaces graphiques Gestion des erreurs et exceptions</p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                                                                                                                                                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h3>Notion de thread</h3> <p>Compléments en P00<br/>Aider Degorre</p> <p>Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Géométrie Concurrence et synchronization les threads Théorie en Java Théorie en C/C++ avec les threads Interfaces graphiques Gestion des erreurs et exceptions</p> | <p><b>Alors</b></p> <pre>public class Example {     public static void main(String[] args) {         new ReciteAlphabet().start();         new ReciteAlphabet().start();     } }</pre> <p>peut afficher</p> <p><b>Exemple simple (1)</b></p> <pre>public class Example {     public static void main(String[] args) {         new ReciteAlphabet().start();         new ReciteAlphabet().start();     } }</pre> <p><b>Exemple de deux threads, l'un qui compte jusqu'à 10 alors que l'autre récite l'alphabet :</b></p> <pre>class ReciteAlphabet extends Thread {     @Override     public void run() {         for (int i = 0; i &lt; 10; i++) {             System.out.print(i + " ");         }     } }  class Example {     public void run() {         for (int i = 0; i &lt; 26; i++) {             System.out.print((char)(i + 'a'));         }     } }</pre> <p><b>Exemple</b></p> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

<sup>374.</sup> On parle typiquement de milliers, pas de millions. La limite pratique est la mémoire disponible.

<sup>375.</sup> réels ou simulés, cf. hyperthreading

<sup>376.</sup> Contexte = pointeur de pile, pointeur ordinal, différents registres...

## Notion de *thread*

Compléments en P00  
Aidez Degorre

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence concurrence et décomposition de l'espace Thread en jeu Threadpool Java Interfacing graphiques Gestion des erreurs et exceptions

Dans le **runtime des langages de programmation** : des langages de programmation (*Erlang*, *Go*, *Haskell*, *Lua*, ...), mais pas actuellement Java), contiennent une notion de *thread* « léger » (différents noms : *green thread*, *fibre*, *coroutine*, *goroutine*, ...), s'exécutant par-dessus un ou des *threads* système.

**OS (noyau)**

|                 |                                                                     |               |               |               |
|-----------------|---------------------------------------------------------------------|---------------|---------------|---------------|
| Langage/runtime | Abstractions (fibres, coroutines, acteurs, futurs, événements, ...) |               |               |               |
|                 | <i>thread</i>                                                       | <i>thread</i> | <i>thread</i> | <i>thread</i> |
|                 | Ordonnanceur                                                        |               |               |               |
| Matériel        | Proc. logique                                                       | Proc. logique | Proc. logique | SMT           |
|                 | SMT                                                                 | Cœur          | Cœur          | CPU 388       |

387. Sous entendu : « *thread* système » (*« thread » sans précision = « thread système »*).  
388. Possible aussi : plusieurs CPUs (plusieurs cœurs par CPU, plusieurs processeurs logiques par cœur...)

## À propos des *threads* système

Compléments en P00  
Aidez Degorre

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence concurrence et décomposition de l'espace Thread en jeu Threadpool Java Interfacing graphiques Gestion des erreurs et exceptions

Ce sont des *threads*.

**Avantage** : se programme séquentiellement (respectent les habitudes),  
**Inconvénient** : la synchronisation doit être explicitée par le programmeur.  
389

**Multi-tâche préemptif** : l'ordonnanceur peut suspendre un *thread* (au profit d'un autre), à tout moment

**Avantage** : pas besoin de signaler quand le programme doit « laisser la main ».  
**Inconvénient** : changements de contexte fréquents et coûteux.

**Implémentation dans le noyau** :

- **Implémentation dans le noyau** :
- **Avantage** : compatible avec tous les exécutables de l'OS (pas seulement JVM)
- **Inconvénient** : fonctionnalités rudimentaires. P. ex., chaque *thread* à une pile de taille fixe (1024 Ko pour les *threads* de la JVM 64bits) → peu économique!

389. On va voir dans la suite comment. Pour l'instant, retenez qu'il n'y a aucune synchronisation, donc aucun partage de données sur entre *threads* si on n'ajoute pas « quelque chose ».

## Des APIs de haut niveau pour ne pas manipuler les *threads*

Compléments en P00  
Aidez Degorre

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence concurrence et décomposition de l'espace Thread en jeu Threadpool Java Interfacing graphiques Gestion des erreurs et exceptions

→ les langages de programmation proposent des mécanismes, utilisant les *threads* système, pour pallier leurs inconvénients tout en essayant<sup>390</sup> de garder leur avantages.

Au moins deux objectifs :

- limiter le nombre de *threads* système utilisés, afin de diminuer l'empreinte mémoire et la fréquence des changements de contexte
- forcer des procédés sûrs pour le partage de données ; ou à défaut, faciliter les bonnes pratiques de synchronisation.

390. avec plus ou moins de succès

## Threads en Java

Compléments en P00  
Aidez Degorre

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence concurrence et décomposition de l'espace Thread en jeu Threadpool Java Interfacing graphiques Gestion des erreurs et exceptions

1 *thread* est associé à 1 pile d'appel de méthodes.

*Thread* principal en Java = pile = pile des méthodes appelées depuis l'appel initial à *main()*

→ vous utilisez déjà des *threads* !

- **Interfaces graphiques (Swing, JavaFX, ...)** : un *thread* ( $\neq \text{main}$ ) est dédié aux évènements de l'IG :
- Programmation événementielle → méthodes gestionnaires d'évènement
- Évènements → mis en file tâche quand ils surviennent.
- Quand le *thread* des évènements est libre, le gestionnaire correspondant au premier évènement de la file est appellé et exécuté sur ce *thread*.

**Intérêt** : 395 pas besoin de prévoir des interruptions régulières dans le *thread main* pour vérifier et traiter les événements en attente (l'IG resterait figé entre deux).

394. Pour Swing : Event Dispatching Thread (EDT). Pour JavaFX : JavaFX Application Thread.

395. Et l'intérêt de faire qu'un seul *thread* pour cela : la sûreté du fonctionnement de l'IG. Pas d'entreclacs entre 2 événements, pas d'accès compétition.

## Threads en Java

Compléments en P00  
Aidez Degorre

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence concurrence et décomposition de l'espace Thread en jeu Threadpool Java Interfacing graphiques Gestion des erreurs et exceptions

utilise directement les *threads* système, via la classe *Thread*.

à historiquement (Java 1.1) utilisé des *green threads*<sup>391</sup>, abandonnés pour des raisons de performance<sup>392</sup>.

- pourrait néanmoins, dans le futur, supporter les fibres<sup>393</sup> via le projet Loom.
- dispose actuellement d'un grand nombre d'APIs facilitant où rendant plus sûre l'utilisation des *threads* : les boucles d'évènements Swing et JavaFX, ThreadPoolExecutor, ForkJoinPool/ForkJoinTask, CompletableFutureFuture, Stream...

391. Une sorte de *threads* légers.

392. Ils étaient ordonnances sur un seul *thread* système, empêchant d'utiliser plusieurs processeurs.

393. Autre type de *threads* légers. Cette fois-ci le travail peut être distribué sur plusieurs *threads* système. Des implémentations de fibres pour Java existent déjà : bibliothèques Quasar et Klimt. Mais pour fonctionner, celles-ci doivent modifier le code-objet généré par *javac*.

## Créer un *thread* en Java

Compléments en P00  
Aidez Degorre

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence concurrence et décomposition de l'espace Thread en jeu Threadpool Java Interfacing graphiques Gestion des erreurs et exceptions

Définir et instancier une classe héritant de la classe *Thread* :

```
public class HelloRunnable extends Thread {
    @Override
    public void run() {
        System.out.println("Hello, from a thread!");
    }
}
```

// plus loin

```
HelloThread().start();
```

Implémenter *Runnable* et appeler le constructeur *Thread(Runnable target)* :

```
public class HelloRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("Hello, from a thread!");
    }
}
```

// plus loin

```
new Thread(new HelloRunnable()).start();
```

Mais pour un *thread* simple, on préférera écrire une lambda-expression :

```
new Thread(() -> {
    System.out.println("Hello, from a thread!");
}).start();
```

## La classe Thread

Compléments en P00  
Aidez Degorre

Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence concurrence et décomposition de l'espace Thread en jeu Threadpool Java Interfacing graphiques Gestion des erreurs et exceptions

String *getName()* : récupérer le nom d'un *thread*.

*void join()* : attendre la fin de ce *thread* (voir synchronisation).

*void run()* : la méthode qui lance tout le travail de ce *Thread*. C'est la méthode qu'il faudra redéfinir à chaque fois que *Thread* sera étendue !.

**static void sleep(long millis)** : met le *thread* courant (i.e. en cours d'exécution) en pause pendant tant de ms. (NB : c'est une méthode **static**). Le *thread* mis en pause est celui qui appelle la méthode. Il n'y a pas de **this**!.

**void start()** : démarre le *thread* conséquence : *run()* est exécutée dans le nouveau thread : celui décrété par l'objet, pas celui de l'appelant!.

**void interrupt()** : interrompt le *thread* (rééchappe **InterruptedException** si le *thread* était en attente sur *wait()*, *join()*, *sleep()*...).

**static boolean interrupted()** : teste si un autre *thread* a demandé l'interruption du *thread* courant.

396. => *Thread* est ainsi un décorateur de *Runnable*.

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

Une instance de `thread` est toujours dans un des états suivants :

- `NEW`: juste créé, pas encore démarré.
  - `RUNNABLE`: en cours d'exécution.
  - `BLOCKED`: en attente de moniteur (voir la suite).
  - `WAITING`: en attente d'une condition d'un autre `thread` (voir `notify()`/`wait()`).
  - `TIME_WAITING`: idem pour attendre avec temps limite.
  - `TERMINATED`: exécution terminée.
- Mais attends la suite pour en dire plus sur ces états...

- `Si t est un thread, l'appel t.interrupt() demande l'interruption de celui-ci.`
- `Si t est en train d'exécuter une méthode interruptible397, celle-ci quitte tout de suite.`

`L'interruption est propagée le long des méthodes de la pile d'appel qui quittent une à une... jusqu'à la méthode principale de la tâche398, qui quitte aussi.`

`Le résultat (non garanti399) est la terminaison de la tâche exécutée sur t400.`

`La propagation de l'interruption est implémentée par la propagation de l'exception InterruptedException et par le contrôle du boolean Thread.interrupted()401 (détails juste après).`

`397. C'est le cas de toutes les méthodes bloquantes de l'API Thread : wait(), sleep(), join() ...`

`398. Habituellement : run.`

`399. Si les méthodes exécutées sur t n'ont pas prévu d'être interrompus, rien ne se passe.`

`400. Si l'exécution directe dans le thread, terminaison du thread, sinon, si l'exécution dans un thread pool, l'interruption a eu lieu. 2 cas de figure (voir la suite).`

Pour écrire une méthode interruptible f :

- Quand une interruption est détectée la bonne pratique est de quitter (`return` ou `throw`) au plus tôt, tout en libérant les ressources utilisées.
- L'interruption peut être détectée de deux façons :
  - soit une méthode auxiliaire g appelée depuis f, quitte sur InterruptedException
  - soit on a obtenu true en appelant Thread.interrupted().
- Le premier cas (exception) doit être traité en mettant tout appelle à g dans un block try/finally (libération explicite des ressources de f dans le finally) ou bien try-with-resource (libération implicite).
- Remarque : il faut absolument vérifier Thread.interrupted() dans toute boucle de f ne faisant pas d'appel à une méthode interruptible comme g.
- Dans tous les cas, il faut veiller à propager le statut « interrompu » au contexte d'exécution, pour qu'il puisse, lui aussi, prendre en compte le fait qu'une interruption a eu lieu. 2 cas de figure (voir la suite).

## Problèmes liés au multithreading

Introductions  
Compléments en P40  
Autre-Dégorge

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

Deux principaux problèmes :

- 1 **Les entrelacements non maîtrisés** : les instructions de 2 threads s'entrelacent et accèdent (lecture et écriture) aux mêmes données dans un ordre imprévisible. Ce phénomène est « naturel » (l'ordonnanceur est libre de faire avancer un thread, puis l'autre au moment où il veut), il est parfois gênant, parfois non.

- 2 **Les incohérences dues aux optimisations matérielles**<sup>402</sup> : la JVM 403 laisse une marge d'interprétation assez large au matériel pour qu'il puisse exécuter le programme efficacement. Principales conséquences :
- ordre des instructions donné dans le code source pas forcément respecté
  - modifications de variables partagées pas forcément vues par les autres threads.

Pour l'instant, concentrons nous sur le problème 1.

401. `interrupt()`

402. En particulier dans le microprocesseur

403. La JVM s'appuie sur le **JMM** : Java Memory Model, un module d'exécution relativement lax.

## Accès atomique

Introductions  
Compléments en P40  
Autre-Dégorge

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

## La classe Thread

Introductions  
Compléments en P40  
Autre-Dégorge

Avec quelle granularité les entrelacements se font-ils ? Peut-on s'arrêter au milieu d'une affectation, faire autre chose sur la même variable, puis finir → notion clé **atomicité**

- **Atomique** = non séparable (éym.) non entrelaçable (ici).
- Aucune autre instruction, accédant aux mêmes données, ne peut être exécutée pendant celle des instructions d'une opération atomique.
- Quelques exemples d'opérations atomiques :
  - lecture ou affectation de valeur 32 bits (`boolean, char, byte, short, int, float`);
  - lecture ou affectation d'un attribut `volatile`<sup>404</sup>;
  - lecture ou affectation d'un bloc `synchronized`<sup>405</sup> ;
  - exécution d'un bloc `finally` ;

- Exemple d'opération non atomique : `x++` (peut se décomposer ainsi : copie x en pile, empile 1, additionne, copie le sommet de pile dans x).
- 404. Notion abordée plus loin.
- 405. Idem. Dans ce cas, remplacer « accédant aux mêmes données » par « utilisant le même verrou ».

Contrairement à ce qu'on pourrait attendre, les nombres ne sont pas dans l'ordre, certains se répètent, d'autres n'apparaissent pas.

Exécution possible :

```
    x incrémenté par t2, sa nouvelle valeur est 2.
    x incrémenté par t1, sa nouvelle valeur est 2.
    x incrémenté par t2, sa nouvelle valeur est 3.
    x incrémenté par t1, sa nouvelle valeur est 4.
    x incrémenté par t2, sa nouvelle valeur est 5.
    x incrémenté par t1, sa nouvelle valeur est 6.
    x incrémenté par t2, sa nouvelle valeur est 7.
    x incrémenté par t2, sa nouvelle valeur est 9.
    x incrémenté par t2, sa nouvelle valeur est 10.
    x incrémenté par t1, sa nouvelle valeur est 8.
```

Exemple à voir tous les entier de 1 à 10 s'afficher dans l'ordre.

Exemple

## La synchronisation

### Avec la classe Thread : méthode join()

## Synchronisation avec join()

**Synchronisation :**

- consiste, pour un *thread*, à attendre le « feu vert » d'un autre *thread* avant de continuer son exécution;
- interdit certains entrelacements;
- contribue à établir la relation "arrivé-avant", limitant les optimisations autorisées<sup>406</sup>.

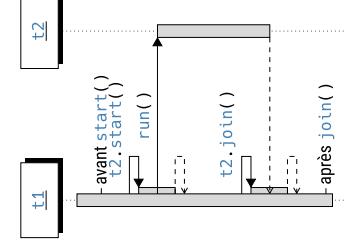
Ici, *a*<sub>1</sub> arrive avant *a*<sub>2</sub>, *b*<sub>1</sub> avant *b*<sub>2</sub>, *a*<sub>1</sub> avant *b*<sub>1</sub>,  
mais pas *b*<sub>1</sub> avant *a*<sub>2</sub> !

406. À suivre...

**Synchronisation simple : attendre la terminaison d'un thread avec join() 407 :**

```
public class ThreadJoinExample {
    public static void main(String[] args) {
        Thread t = new ThreadJoin();
        t.start();
        t.join();
        System.out.println("x");
    }
}
```

affiche ① alors que le même code sans l'appel à *join()* affichera probablement ②.  
Tout ce qui est exécuté dans le *thread* t arrive-avant ce qui suit le *join()* dans le *thread* main (ici, l'incrémentation de *x*).  
407. Existe aussi en version temporisée : on bloque jusqu'au délai donné en paramètre maximum.



**Le verrou intrinsèque**

Qu'est-ce à quoi cela sert-il?

En Java tout objet contient un **verrou intrinsèque** (ou **moniteur**).

- À tout moment, le moniteur est soit libre, soit détenu par un (seul) *thread* donné.
- Ainsi un moniteur met en œuvre le principe d'exclusion mutuelle.
- Lors de son exécution, un *thread* t peut demander à prendre un moniteur.
- Si le moniteur est libre, alors il est pris par t, qui continue son exécution.
- Si le moniteur est déjà pris, t est alors mis en attente jusqu'à ce que le moniteur se libère pour lui (il peut y avoir une liste d'attente).
- Un *thread* peut à tout moment libérer un moniteur qu'il possède.

**Conséquence :** tout ce qui se produit dans un *thread* avant qu'il libère un moniteur arrive-avant ce qui se produit dans le prochain *thread* qui obtiendra le moniteur, après l'obtention de celui-ci.

**Le verrou intrinsèque**

Utilisation en pratique : le mot-clé synchronized

**Bloc synchronisé :**

```
class AutreCompteur {
    private int valeur;
    private Object verrou = new Object(); // peu importe le type déclaré
    public void inc() { // <-- ici !
        synchronized(verrou) { // <-- valeur++;
        }
    }
}
```

**Méthode synchronisée :** cas particulier avec synchronisation de tout le corps de la méthode sur moniteur de **this**. → syntaxe plus légère, plus souvent utilisée en pratique.

équivaut à ...

```
class Compteur {
    private int valeur;
    // méthode contenant bloc synchronisé
    // sur this
    public void incr() {
        synchronized(this) { valeur++; }
    }
}
```

Utilisation en pratique : le mot-clé synchronized

**Le verrou intrinsèque**

Utilisation en pratique : le mot-clé synchronized

**Bloc synchronisé :**

```
class AutreCompteur {
    private int valeur;
    private Object verrou = new Object(); // peu importe le type déclaré
    public void inc() { // <-- ici !
        synchronized(verrou) { // <-- valeur++;
        }
    }
}
```

**Méthode synchronisée :** cas particulier avec synchronisation de tout le corps de la méthode sur moniteur de **this**. → syntaxe plus légère, plus souvent utilisée en pratique.

équivaut à ...

```
class Compteur {
    private int valeur;
    // méthode contenant bloc synchronisé
    // sur this
    public void incr() {
        synchronized(this) { valeur++; }
    }
}
```

Le mécanisme notifyAll()/notify() - conseil d'utilisation

**Le verrou intrinsèque**

Le mécanisme notifyAll()/notify() - conseil d'utilisation

3 méthodes concernées (classe *Object*) : notify(), notifyAll() et wait().

- Ces méthodes sont appelables seulement dans un bloc synchronisé sur l'objet récepteur de l'appel : **synchronized**(x){ x.notify(); }.
- wait() : met le *thread* en sommeil et libère le moniteur (getState() passe de RUNNABLE à WAITING).

Le *thread* restera dans cet état tant qu'il n'est pas réveillé (par **notifyAll()** ou **notify()**). Il sera alors en attente pour récupérer le moniteur (WAITING → BLOCKED).

- notifyAll() : réveille tous les threads en attente sur l'objet. Ceux-ci deviennent candidats à reprendre le moniteur quand il sera libéré.
- notify() : réveille un *thread* en attente sur l'objet.

**Le verrou intrinsèque**

Utilisation en pratique : le mot-clé synchronized

**Méthode synchronisée :** cas particulier avec synchronisation de tout le corps de la méthode sur moniteur de **this**. → syntaxe plus légère, plus souvent utilisée en pratique.

équivaut à ...

```
class Compteur {
    private int valeur;
    // méthode contenant bloc synchronisé
    // sur this
    public void incr() {
        synchronized(this) { valeur++; }
    }
}
```

**Le verrou intrinsèque**

Utilisation en pratique : le mot-clé synchronized

**Méthode synchronisée :** cas particulier avec synchronisation de tout le corps de la méthode sur moniteur de **this**. → syntaxe plus légère, plus souvent utilisée en pratique.

équivaut à ...

```
class Compteur {
    private int valeur;
    // méthode contenant bloc synchronisé
    // sur this
    public void incr() {
        synchronized(this) { valeur++; }
    }
}
```

Le mécanisme notifyAll()/notify() - conseil d'utilisation

**Le verrou intrinsèque**

Utilisation en pratique : le mot-clé synchronized

**Bloc synchronisé :**

```
class AutreCompteur {
    private int valeur;
    private Object verrou = new Object(); // peu importe le type déclaré
    public void inc() { // <-- ici !
        synchronized(verrou) { // <-- valeur++;
        }
    }
}
```

**Méthode synchronisée :** cas particulier avec synchronisation de tout le corps de la méthode sur moniteur de **this**. → syntaxe plus légère, plus souvent utilisée en pratique.

équivaut à ...

```
class Compteur {
    private int valeur;
    // méthode contenant bloc synchronisé
    // sur this
    public void incr() {
        synchronized(this) { valeur++; }
    }
}
```

**Le verrou intrinsèque**

Utilisation en pratique : le mot-clé synchronized

**Bloc synchronisé :**

```
class AutreCompteur {
    private int valeur;
    private Object verrou = new Object(); // peu importe le type déclaré
    public void inc() { // <-- ici !
        synchronized(verrou) { // <-- valeur++;
        }
    }
}
```

**Méthode synchronisée :** cas particulier avec synchronisation de tout le corps de la méthode sur moniteur de **this**. → syntaxe plus légère, plus souvent utilisée en pratique.

équivaut à ...

```
class Compteur {
    private int valeur;
    // méthode contenant bloc synchronisé
    // sur this
    public void incr() {
        synchronized(this) { valeur++; }
    }
}
```

Le mécanisme notifyAll()/notify() - conseil d'utilisation

**Le verrou intrinsèque**

Le mécanisme notifyAll()/notify() - conseil d'utilisation

3 méthodes concernées (classe *Object*) : notify(), notifyAll() et wait().

- On utilise wait() pour attendre une condition **cond**.
- Mais plusieurs threads peuvent être en attente. Un autre pourrait être libéré et récupérer le moniteur avant, rendant la condition à nouveau fausse.
- aucune garantie que **cond** soit vraie au retour de **wait()**.

Ainsi, il faut tester à nouveau jusqu'à satisfaire la condition :

```
synchronized(obj) { // conseil : mettre wait dans un while
    while (!condition(obj)) obj.wait();
    ...
    // inserer tct instructions qui avaient besoin de condition()
}
```

Il faut absolument retenir la formule ci-dessus !!!

(utilisée dans 99,9% des cas d'usage corrects de **wait()**...)

**Le verrou intrinsèque**

Le mécanisme notifyAll()/notify() - conseil d'utilisation

3 méthodes concernées (classe *Object*) : notify(), notifyAll() et wait().

- On utilise wait() pour attendre une condition **cond**.
- Mais plusieurs threads peuvent être en attente. Un autre pourrait être libéré et récupérer le moniteur avant, rendant la condition à nouveau fausse.
- aucune garantie que **cond** soit vraie au retour de **wait()**.

Ainsi, il faut tester à nouveau jusqu'à satisfaire la condition :

```
synchronized(obj) { // conseil : mettre wait dans un while
    while (!condition(obj)) obj.wait();
    ...
    // inserer tct instructions qui avaient besoin de condition()
}
```

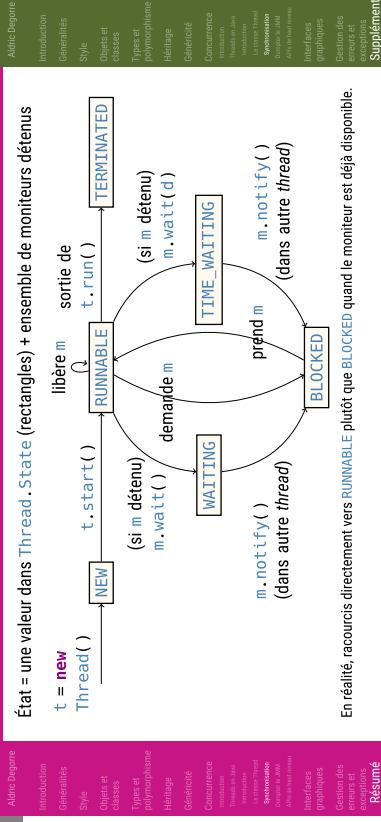
Il faut absolument retenir la formule ci-dessus !!!

(utilisée dans 99,9% des cas d'usage corrects de **wait()**...)

## Le verrou intrinsèque

| Compléments en P00                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Aidez Degorre                                                                                                                                                                                            | Mécanisme <code>not(if(y))y()  y().wait();</code> – alternatives                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence threadsafe Threadsafe et Java Interficies et graphiques Gestion des erreurs et exceptions Résumé | <p>Le mécanisme <code>not(if(y))y()  y().wait();</code> – alternatives</p> <ul style="list-style-type: none"> <li><b>Variantes acceptables :</b> <ul style="list-style-type: none"> <li><b>while(i &lt; condition()) Thread.sleep(temp);</b> → utilise quand on sait qu'aucun <code>thread</code> ne notifiera quand la condition sera vraie.</li> <li><b>while(i &lt; condition()) Thread.onSpinWait(); ; (Java ≥ 9) : attente active</b> (c'est-à-dire : ni blocage ni attente, le <code>thread</code> teste <b>RUNNABLE</b>).</li> </ul> </li> <li>→ on évite le coût de la mise en attente et du réveil, cette approche est donc conseillée quand on s'attend à ce que la condition soit vraie très vite.</li> </ul> <p><b>Déconseillé 408 :</b> <code>while(i &lt; condition()) obj.wait(); ;</code> : attente active « bête » → c'est l'ancienne façon de faire, remplacée avantagéusement par la variante avec <code>onSpinWait()</code>. En effet, <code>onSpinWait</code> signale à l'ordonnanceur que le <code>thread</code> peut être mis en pause (laisser sa place sur le processeur) prioritairement en cas de besoin.</p> <p>408. Sauf pour faire cuire une omlette sur son microprocesseur...</p> |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence threadsafe Threadsafe et Java Interficies et graphiques Gestion des erreurs et exceptions Résumé |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

## Retour sur les états d'un thread



## Verrous explicites (1)

| Compléments en P00                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Aidez Degorre                                                                                                                                                                                                       | <p><b>Etat</b> = une valeur dans <code>Thread.State</code> (rectangles) + ensemble de moniteurs détenus</p> <pre> graph LR     NEW[NEW] -- "t = new Thread()" --&gt; RUNNABLE[RUNNABLE]     RUNNABLE -- "t.start()" --&gt; RUNNABLE     RUNNABLE -- "m.wait()" --&gt; WAITING[WAITING]     WAITING -- "m.notify()" --&gt; RUNNABLE     WAITING -- "m.notify()" --&gt; BLOCKED[BLOCKED]     BLOCKED -- "m.wait()" --&gt; WAITING     BLOCKED -- "m.notify()" --&gt; RUNNABLE     </pre> <p>En réalité, racourci directement vers <b>RUNNABLE</b> pluriel que <b>BLOCKED</b> quand le moniteur est déjà disponible.</p>                                                                                                                                                                                                                                                                    |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence threadsafe Threadsafe et Java Interficies et graphiques Gestion des erreurs et exceptions Résumé Supplément | <ul style="list-style-type: none"> <li>moniteurs = principe d'exclusion mutuelle + mécanisme d'attente/notification;</li> <li>mais il existe d'autres façons de synchroniser des <code>threads</code> par rapport à l'usage d'une ressource (exemple : lecteur/rédacteur);</li> <li>fonctionnalités possibles : savoir à qui appartient le verrou, qui est en attente, etc. ;</li> <li>→ bibliothèque de verrous divers dans <code>java.util.locks.Lock</code>, implémentant l'interface <code>java.util.concurrent.locks.Lock</code>.</li> </ul> <p>L'interface <code>java.util.concurrent.locks.Lock</code> :</p> <pre> public interface Lock {     void lock();     void lockInterruptibly() throws InterruptedException;     Condition newCondition();     boolean tryLock();     boolean tryLock(long time, TimeUnit unit) throws InterruptedException;     void unlock(); } </pre> |

## Dangers de la synchronisation

| Compléments en P00                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Aidez Degorre                                                                                                                                                                                            | <p><b>Dangers de la synchronisation</b></p> <p>quand elle est utilisée à mauvais escient ou à l'envers</p> <p><b>Un dernier avertissement :</b> la synchronisation doit rester raisonnable !</p> <p>En général, plus il y a de synchronisation, moins il y a de parallélisme... et plus le programme est talenté. Pire, il peut bloquer.</p> <p><b>Pathologies typiques :</b></p> <ul style="list-style-type: none"> <li><b>dead-lock</b> : 2 threads attendent chacun une ressources que seul l'autre serait à même de libérer (en fait 2 ou plus : dès lors que la dépendance est cyclique).</li> <li><b>famine</b> (starvation) : une ressource est réservée trop souvent/trop longtemps toujours par la même tâche, empêchant les autres de progresser.</li> <li><b>live-lock</b> : boucle infinie cause par plusieurs threads se faisant réagir mutuellement, sans pour autant faire avancer le programme. 411</li> </ul> <p>411. Simaginer deux individus essayant de se croiser dans un couloir entamant simultanément une manœuvre d'évitement du même côté, mettant les deux personnes à nouveau l'une face à l'autre, provoquant une nouvelle manœuvre d'évitement, et ainsi de suite...</p> |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence threadsafe Threadsafe et Java Interficies et graphiques Gestion des erreurs et exceptions Résumé |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

## Verrous explicites (2)

| Compléments en P00                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Aidez Degorre                                                                                                                                                                                                       | <p><b>Comme le verrouillage et le déverrouillage se font par appels explicites aux méthodes Lock et un Lock, ces verrous sont appelés <b>verrous explicites</b>.</b></p> <p><b>Inconvénient :</b> l'occupation du verrou n'est pas délimitée par un bloc lexical tel que <b>synchronized { ... }</b>.</p> <p>La logique du programme doit assurer que toute exécution de <code>lock</code> soit suivie d'une exécution de <code>unlock</code>.</p> <p><b>Avantages :</b></p> <ul style="list-style-type: none"> <li>NOMBREUSES options de configuration.</li> <li>Flexibilité dans l'ordre d'acquisition et de libération. 410</li> </ul> <p>410. Mais on peut programmer un tel bloc à la main à l'aide d'une fonction d'ordre supérieur, et encapsuler un tel verrou dans une classe dont l'interface ne permettrait d'accéder le verrou que via cette FOS.</p> <p>410. Concurrent Programming in Java (2.1.4) montre un exemple de liste chaîne concurrente où, lors d'un parcours, il est nécessaire d'exécuter une chaîne d'acquisitions/libérations de la forme :</p> <pre> m.lock(); ... ; m.unlock(); ... ; m.lock(); ... ; m.unlock(); ... </pre> |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence threadsafe Threadsafe et Java Interficies et graphiques Gestion des erreurs et exceptions Résumé Supplément |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Éviter les dead-locks

| Compléments en P00                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Aidez Degorre                                                                                                                                                                                                       | <p>À l'aide d'une utilisation raisonnée des verrous</p> <ul style="list-style-type: none"> <li>Principe pour éviter les <code>deadlocks</code> : <b>toujours acquérir les verrous dans le même ordre</b> 412 et les libérer dans l'ordre inverse 413 (ordre LIFO, donc).</li> <li>En effet : dans l'exemple précédent, une exécution de <code>t.un</code> veut acquérir <code>o1</code> puis <code>o2</code>, alors que l'autre exécution veut faire dans l'autre sens.</li> <li>→ quand on écrit un programme concurrent à l'aide de verrous explicites, il faut documenter un ordre unique pour prendre les verrous.</li> </ul> <p>L'autre voie est de se reposer sur des abstractions de plus haut niveau, sur lesquelles il est plus aisée de raisonneur (cf. la suite).</p> |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence threadsafe Threadsafe et Java Interficies et graphiques Gestion des erreurs et exceptions Résumé Supplément |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

## Paradigme des threads

| Compléments en P00                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Aidez Degorre                                                                                                                                                                                                    | <p><b>Est-ce la réalité ?</b></p> <ul style="list-style-type: none"> <li>Rappel : <code>thread</code> = abstraction</li> <li>simulant la <u>sequentialité</u> dans le <code>thread</code>,</li> <li>permettant une communication instantanée inter<code>thread</code> via une <u>mémoire partagée</u>,</li> <li>(et simulant le <u>parallelisme partagé</u> 414 entre <code>threads</code>).</li> </ul> <p>Est-ce vraiment la réalité ?</p> <p>→ Divulgachage : <b>NON</b> !</p> <p>En réalité, paradigme idéal trop contraintant, empêchant les optimisations matérielles.</p> <p>Modèle d'exécution réellement implémenté par la JVM : le <b>JMM</b> 415.</p> <p>Seule garantie sous condition, ce qu'on observe est indistinguable du paradigme idéal.</p> <p>414. Hors synchronisation, évidemment.</p> <p>415. Java Memory Model.</p> |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence threadsafe Threadsafe et Java Interficies et graphiques Gestion des erreurs et exceptions Résumé Analyse |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

## Modèle de mémoire Java et optimisations

| Compléments en P00                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Aidez Degorre                                                                                                                                                                                                    | <p>Cohérence de la mémoire</p> <p><b>Réalité physique</b> : chaque cœur de CPU dispose de son propre cache 416 de mémoire.</p> <p><b>Interprétation</b> : Chaque <code>thread</code> utilise potentiellement un cache de mémoire différent. Ainsi, les données partagées existent en de multiples copies pas forcément à jour (on parle de problèmes de <u>visibilité</u> des changements et de <u>cohérence</u> de la mémoire).</p> <p><b>Solution naïve</b> : répercuter immédiatement les changements dans tous les caches immédiatement.</p> <p>→ <b>le JMM</b> : ne garantit donc pas une cohérence parfaite (mais un minimum quand-même...) quand-même.)</p> <p>Problème : cette opération est coûteuse et supprime le bénéfice du cache.</p> |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence threadsafe Threadsafe et Java Interficies et graphiques Gestion des erreurs et exceptions Résumé Analyse |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## 416. Mémoire locale propre au cœur, plus proche physiquement et plus rapide que la mémoire centrale.

| Compléments en P00                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Aidez Degorre                                                                                                                                                                                                    | <p>Cohérence de la mémoire</p> <p><b>Réalité physique</b> : chaque cœur de CPU dispose de son propre cache 416 de mémoire.</p> <p><b>Interprétation</b> : Chaque <code>thread</code> utilise potentiellement un cache de mémoire différent. Ainsi, les données partagées existent en de multiples copies pas forcément à jour (on parle de problèmes de <u>visibilité</u> des changements et de <u>cohérence</u> de la mémoire).</p> <p><b>Solution naïve</b> : répercuter immédiatement les changements dans tous les caches immédiatement.</p> <p>→ <b>le JMM</b> : ne garantit donc pas une cohérence parfaite (mais un minimum quand-même...) quand-même.)</p> <p>Problème : cette opération est coûteuse et supprime le bénéfice du cache.</p> |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence threadsafe Threadsafe et Java Interficies et graphiques Gestion des erreurs et exceptions Résumé Analyse |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

## 417. Mémoire locale propre au cœur, plus proche physiquement et plus rapide que la mémoire centrale.

### Par exemple, dans le programme suivant :

```
public class ThreadTest {
    static boolean x = false, y = false;
    public void run() {
        if (x) { x = true; } else System.out.println("Bug !");
        // Affiche "Bug !" si on trouve y vrai alors que x est faux
        public static void test(int nthreads) throws InterruptedException {
            for (int i = 0; i < nthreads; i++) new ThreadConsistency().start();
        }
    }
}
```

L'appel `test(100)` peut afficher « **Bug !** ». 417  
Par exemple : si un des threads finit d'exécuter `x = true; y = true;` et un autre `reçoit`, dans son cache, la modification sur `y` mais pas sur `x`.  
417. Le JMM autorise cette possibilité théorique, mais probablement vous ne verrez jamais ce message !

Exemple

**Réalité physique** : Les CPU sont dotés de mécanismes permettant de réordonner des instructions<sup>418</sup> qu'il sait devoir exécuter (afin de mieux occuper tous ses composants).

**Interprétation** : l'ordre du programme n'est pas toujours respecté (même sur 1 thread). En plus, optimisations différentes d'une architecture matérielle à une autre (p. ex. comportement différent entre x86 et ARM) → ordre peu prévisible.

**Solution naïve** : ajouter des barrières<sup>419</sup> partout dans le code compilé.

**Problème** : vitesse d'exécution sous-optimale (le CPU l'arrive plus à donner autant de travail à tous ses composants).

→ **le JMM** : ne garantit pas le respect exact de l'ordre du programme... mais promet que certaines choses importantes restent bien ordonnées.

418. out-of-order execution  
419. Instruction spécifique prévue dans les CPU, justement pour empêcher le réordonnement.

### Modèle de mémoire Java et optimisations Principe informel du JMM

Compléments en P40  
Aidez Degorre

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Héritage  
Généricité  
Concurrency  
Interfaces graphiques  
Gestion des erreurs et exceptions

Supplément

Compléments en P40  
Aidez Degorre

Exemples

Entre 2 points de synchronisation, toute optimisation est autorisée tant qu'elle ne change pas le comportement observable<sup>420</sup> d'un *thread* qui s'exécutera sans interférence d'un autre *thread*.

Donc

- *mono-thread* → aucune différence visible due à ces optimisations ;
- mais *multi-thread* → différences possibles si synchronisation insuffisante.

→ au programmeur de faire en sorte d'avoir une synchronisation suffisante afin que ces optimisations ne soient pas un problème.

**Remarque** : il reste à définir précisément ce qu'on entend par interférence et synchronisation suffisante.

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Héritage  
Généricité  
Concurrency  
Interfaces graphiques  
Gestion des erreurs et exceptions

Supplément

Compléments en P40  
Aidez Degorre

Exemples

Introduction  
Généralités

- Supposons qu'à l'initialisation `x == 0` & `y == 0`. On veut exécuter :
  - sur le *thread* 1 : `x = 1`; `y = 1`;
  - et, sur le *thread* 2 : `x = y`; `y = 2`;
- (1) arrive-avant (2) et (3) avant (4)
- Or c'est pourtant possible !

En effet, sur chaque *thread* isolé, inverser les 2 instructions ne change pas le résultat. Comme il n'y a pas de synchronisation, rien n'interdit donc ces inversions. Il est donc possible d'exécuter les 4 instructions dans l'ordre suivant :  
`y = 1; x = 2; a = x; b = y;`

On prouvera qu'un programme est correct en raisonnant sur les ordres arrivé-avant de certaines exécutions particulières : les **exécutions séquentiellement cohérentes**<sup>421</sup>.

421. À suivre !

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Héritage  
Généricité  
Concurrency  
Interfaces graphiques  
Gestion des erreurs et exceptions

Supplément

Compléments en P40  
Aidez Degorre

Exemples

Introduction  
Généralités

- **Ordre d'exécution** = réalité objective, non interprétée, de celle-ci.  
Celui-ci est difficile à prévoir, dépendant des optimisations opérées par le CPU. Il ne respecte pas forcément l'ordre du programme, et donc a fortiori, pas non plus l'ordre arrivé-avant d'une exécution donnée.
- **Ordre arrivé-avant** = interprétation idéale de la réalité (qui considère la logique du programme et des synchronisations),  
De très nombreux ordres d'exécution sont possibles pour un même programme. Gestion des erreurs et exceptions
- **Ordre d'exécution** : ordre chronologique réel d'exécution des instructions.  
Il est défini sans ambiguïté et facile à déduire à partir d'un code source et d'une trace d'exécution (p. ex. : depuis un ordre d'exécution).

On prouvera qu'un programme est correct en raisonnant sur les ordres arrivé-avant de certaines exécutions particulières : les **exécutions séquentiellement cohérentes**<sup>421</sup>.

421. À suivre !

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Héritage  
Généricité  
Concurrency  
Interfaces graphiques  
Gestion des erreurs et exceptions

Supplément

Compléments en P40  
Aidez Degorre

Exemples

Introduction  
Généralités

- pour une exécution donnée :
- **Ordre d'exécution** = réalité objective, non interprétée, de celle-ci.  
Celui-ci est difficile à prévoir, dépendant des optimisations opérées par le CPU. Il ne respecte pas forcément l'ordre du programme, et donc a fortiori, pas non plus l'ordre arrivé-avant d'une exécution donnée.
- **Ordre arrivé-avant** = interprétation idéale de la réalité (qui considère la logique du programme et des synchronisations),  
De très nombreux ordres d'exécution sont possibles pour un même programme. Gestion des erreurs et exceptions
- **Ordre d'exécution** : ordre chronologique réel d'exécution des instructions.  
Il est défini sans ambiguïté et facile à déduire à partir d'un code source et d'une trace d'exécution (p. ex. : depuis un ordre d'exécution).

On prouvera qu'un programme est correct en raisonnant sur les ordres arrivé-avant de certaines exécutions particulières : les **exécutions séquentiellement cohérentes**<sup>421</sup>.

421. À suivre !

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Héritage  
Généricité  
Concurrency  
Interfaces graphiques  
Gestion des erreurs et exceptions

Supplément

Compléments en P40  
Aidez Degorre

Exemples

Introduction  
Généralités

- **Exécution séquentiellement cohérente** : exécution
  - qui suit un ordre total,
  - respectant l'ordre du programme,
  - et telle que pour toute lecture d'un emplacement mémoire, la dernière écriture, dans le passé, sur cet emplacement est prise en compte.

⇒ Une exécution séquentiellement cohérente est donc une exécution idéale, intuitive, du programme, non affectée par les réordonnements et incohérences de cache.

Exemple : si `x = 0, x = 1 et printIn(x)` s'exécutent dans cet ordre et qu'entre `x = 1 et printIn(x)` il n'y a pas d'affectation à `x`, alors c'est bien **1** qui s'affiche.

**Programme correctement synchronisé** : se dit d'un programme dont toute exécution séquentiellement cohérente est sans accès en compétition.

422. Mais les attributs de objets référencé peuvent être partagés !

423. C'est-à-dire 2 accès qui ne sont pas reliés par une chaîne de synchronisations et d'ordres imposés par l'ordre des instructions du programme.

Introduction  
Généralités  
Style  
Objets et classes  
Types et polymorphisme  
Héritage  
Généricité  
Concurrency  
Interfaces graphiques  
Gestion des erreurs et exceptions

Supplément

Compléments en P40  
Aidez Degorre

Exemples

Introduction  
Généralités

- **Accès en compétition** :
- **Programme sans accès en compétition** :

public class Echo {
 private int x;
}
public class Competition {
 public static void main(String args[]) {
 Boite b = new Boite();
 new Thread() {
 public synchronized void seX(String x) {
 this.x = x;
 }
 public synchronized void main(String args[]) {
 private static void main(String args[]) {
 new Thread() {
 public void run() {
 System.out.println(b.get());
 }
 }.start();
 }
 }
 }.start();
 }
}

Ici, rien n'impose que la lecture de `b.x` arrive-avant son affectation ou bien contraire.

Exemple

422. Mais les attributs de objets référencé peuvent être partagés !

423. C'est-à-dire 2 accès qui ne sont pas reliés par une chaîne de synchronisations et d'ordres imposés par l'ordre des instructions du programme.

## Modèle d'exécution

La vérité, enfin !

L'ordre d'exécution exact d'un programme est imprévisible, mais ce qui suit est garanti :

- Si le programme est correctement synchronisé<sup>424</sup>, alors son exécution est indiscernable d'une exécution séquentiellement cohérente, donc qu'il n'y ait pas de compétition.

Concrètement, pour que les optimisations ne provoquent pas d'incohérences, il « suffit » de plus dur reste de trouver quelques accès sont en compétition...

Remarque : le plus dur reste de trouver quelques accès sont en compétition... « normales » depuis la propriété ci-dessus, il suffit de vérifier executions visiblement anormale.

Heureusement, d'après la propriété ci-dessus, il suffit de vérifier executions visiblement anormale.

424. Note : si la synchronisation est incorrecte, cela ne veut pas dire qu'on ne sait rien. En fait, la spécification donne tout un ensemble de règles basées sur un critère de causalité... Règles trop compliquées et donnant des garanties trop faibles pour être raisonnablement utilisables en pratique.

Résumé

## Les mots-clés volatile et final (1)

Attributs volatile : un attribut déclaré avec **volatile** garantit<sup>425</sup> :

- que tout accès en lecture se produisant, chronologiquement, après un accès en écriture, arrive-après celui-ci,
- (concrètement : cet attribut n'est jamais mis dans le cache local d'un thread)
- que tout accès simple en lecture ou écriture est atomique (même pour **long** et **double**).

→ comme si cet attribut était accédé via des accesseurs **synchronized**.

Attributs finaux :

- déjà vu : un attribut **final** ne peut être affecté qu'une seule fois (lors de linitialisation de la classe ou de l'objet).
- garantie supplémentaire : comme pour **volatile** tout accès en lecture à un attribut **final** arrive après son initialisation (unique, pour **final**).

425. Ceci ne concerne pas le contenu de l'éventuel objet référence.

## Les objets immuables

Faire des calculs en utilisant les objets immuables

Typiquement, une étape de calcul consiste à créer un nouvel objet immuable à partir d'objets immuables existants (puisque ne peut pas les modifier).

Un tel calcul peut être réalisé à l'aide d'une **fonction pure**<sup>430</sup>.

Inconvénient : implique d'allouer un nouveau objet pour chaque étape de calcul. (coûteux, mais pas forcément excessif<sup>431</sup>)

Le résultat doit être correctement publié pour être utilisable par un autre thread :

- grâce aux mécanismes (méthodes) de passage de message prévues par l'API util.
- ou bien à la main », en l'enregistrant dans une variable partagée (soit **volatile**, soit **private** avec accesseurs **synchronized**) modifiable.

Exemple : **SharedResources.setX()** (si **SharedResources.getX()** ; où **getX** et **setX** sont **synchronized** et **f** est une fonction pure).

430. Fonction sans effet de bord, notamment sans modification d'état persistante.

431. Notamment si l'escape analysis détermine que l'objet n'est que d'usage local → la JVM l'alloue en pile. Cela dit, ne concerne que les calculs intermédiaires car la variable partagée est stockée dans le tas.

## Les mots-clés volatile et final (2)

Technique infaillible : tous les attributs **volatile** (ou **final**) ⇒ accès en compétition impossible. Cependant pas idéale car :

- non réaliste : un programme utilise des classes faites par d'autres personnes,<sup>426</sup>
- non efficace : **volatile** empêche les optimisations ⇒ exécution plus lente.<sup>427</sup>

En plus, **volatile** ne permet pas de rendre les méthodes atomiques ⇒ entrelacements toujours non maîtrisés ⇒ **volatile** ne suffit pas toujours pour tout.

Exemple : avec **volatile int x = 0** ; si on exécute 2 fois en parallèle **x++**, on peut toujours obtenir 1 au lieu de 2.

426. Mais on peut encapsuler leurs instances dans des classes à méthodes synchronisées... au prix d'encore un peu moins de performance.

427. Remarque : pour **final**, la question ne se pose qu'au début de la vie de l'objet. À ce stade, accéder à une ancienne version de l'attribut n'aurait aucun sens. Loptimisation serait nécessairement fausse.

Résumé

## Les objets immuables

Comment éviter les compétitions ?

- éviter de partager les variables quand ce n'est pas nécessaire → privilégier les variables locales (jamais partagées) aux attributs;
- quand ça suffit, privilégier les données partagées en lecture seule → privilégier les structures immuables (voir ci-après);
- sinon, renforcer la relation arrivé-avant :
  - utiliser les mécanismes de synchronisation déjà présentés,
  - marquer des attributs comme **final** ou **volatile** (voir ci-après);
- Souvent, rien de tout ça ne convient : on peut avoir besoin d'attributs modifiables sans synchronisation ! Mais il faudra s'assurer qu'un seul thread peut y accéder.

Autres objets *thread-safe*

## Les objets immuables

Compléments en PdO Autre Degorre

- Rappel : immutable** = non modifiable. Le terme s'applique aux objets et, par extension, aux classes dont les instances sont des objets immuables.
- Ces objets ont généralement des champs tous **final**. Conséquence : relation « arrivé-avant » entre linitialisation de lobjet et tout accès ultérieur.
- Pendant la vie de lobjet : pas d'accès en écriture ⇒ pas d'accès en compétition.
- ⇒ non seulement lutilisation qui en est faite dans un thread n'influe pas sur lutilisation dans un autre thread<sup>428</sup>, mais en plus il ne peut pas y avoir d'incohérence de cache par rapport au contenu d'un objet immuable.<sup>429</sup>

Remarque : tout cela reste vrai quand on parle des champs **final** d'un objet quelconque.

428. Donc tout objet immuable est *thread-safe*.

429. Si lobjet immuable est correctement publié, tous les threads sont d'accord sur lensemble des valeurs publiées.

## Variables atomiques

java.util.concurrent.Atomic\*

- atomiques** (classes mutables *thread-safe*).
- Exemples : **AtomicBoolean**, **AtomicInteger**, **AtomicLong**, ...
- Leurs instances représentent des booleens, des entiers, des tableaux d'entiers, ...
- Accès simples : comportement similaire aux variables volatiles.
- Disposent, en plus, d'opérations plus complexes et malgré tout atomiques (typiquement : incrémentation).<sup>432</sup>

432. L'accès atomique est garantit sans synchronisation, grâce à des appels à des instructions dédiées des processeurs, telles que CAS (compare-and-set). Ainsi ces classes ne sont en réalité pas implémentées en Java, car elles sont compilées en tant que code spécifique à l'architecture physique (celle sur laquelle tourne la JVM).

## Supprimer les compétitions

Compléments en PdO Autre Degorre

- Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Concurrence interne Théorie de base Dernière JMM Interfacing graphiques Gestion des erreurs et exceptions Résumé
- Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Concurrence interne Théorie de base Dernière JMM Interfacing graphiques Gestion des erreurs et exceptions Résumé

Autres objets *thread-safe*

Compléments en PdO Autre Degorre

- Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Concurrence interne Théorie de base Dernière JMM Interfacing graphiques Gestion des erreurs et exceptions Résumé
- Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Concurrence interne Théorie de base Dernière JMM Interfacing graphiques Gestion des erreurs et exceptions Résumé

| Les nombreux inconvénients de l'API <code>threads</code>                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en PdO                                                                                                                                                                                           | Schéma de base, pour limiter le nombre de <code>threads</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence interne Théorie des threads et locks Interfaces graphiques Gestion des erreurs et exceptions Analyse | <p>Utiliser directement les <code>threads</code> et les moniteurs → nombreux inconvénients :</p> <ul style="list-style-type: none"> <li>• Chaque <code>thread</code> utilise beaucoup de mémoire. Et les instancier prend du temps.</li> <li>• Trop de <code>threads</code> → changements de contexte fréquents (opération coûteuse).</li> <li>• Nécessité de communiquer par variables partagées → risque d'accès en compétition (et donc d'incohérences)</li> <li>• En cas de synchronisation mal faite, risque de blocage.</li> </ul> <p><b>Heureusement</b> : de nombreuses API de haut niveau<sup>433</sup> aident à contourner ces écueils. → on travaillera plutôt avec celles-ci que directement avec les <code>threads</code> et les moniteurs.</p> <p><u>433.</u> programmées par dessus les <code>threads</code> et les moniteurs</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Compléments en PdO                                                                                                                                                                                           | <b>APIs pour la concurrence</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence interne Théorie des threads et locks Interfaces graphiques Gestion des erreurs et exceptions Analyse | <p><b>Idée</b> : réutiliser un même <code>thread</code> pour plusieurs exécuter plusieurs tâches tour à tour.</p> <p><b>Exécuteur</b> : objet qui gère un certain ensemble de <code>threads</code></p> <ul style="list-style-type: none"> <li>• en distribuant des <b>tâches</b> sur ceux-ci, selon politique définie ;       <ul style="list-style-type: none"> <li>en évitant de créer plus de <b>threads</b> que nécessaire<sup>434</sup> ;</li> <li>et en évitant de détruire un <code>thread</code> aussitôt qu'il est libre (pour éviter d'en re-créer).</li> </ul> </li> </ul> <p><b>Tâche</b></p> <ul style="list-style-type: none"> <li>• séquence d'instructions (en pratique : une fonction) à exécuter sur un <code>thread</code></li> <li>• ne peut pas être mise en pause pour libérer le <code>thread</code> au profit d'une autre.<sup>435</sup> ↗ ceurs.</li> <li>• 434. Selon politique de l'exécuteur. Plusieurs possibles. Par exemple : nb. max. <code>threads</code> ≤ nb. cœurs.</li> <li>• 435. Le <code>thread</code>, lui, peut être mis en pause par le noyau pour libérer un processeur au profit d'un autre <code>thread</code>.</li> <li>• 436. En principe, car avec <code>ForkJoinPool</code>, notamment, certains appels de méthodes permettent la mise en pause → multi-tâche coopératif.</li> </ul> |
| Compléments en PdO                                                                                                                                                                                           | <b>Interfaces ExecutorService, Callable&lt;V&gt; et Future&lt;V&gt;</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

| APIs pour la concurrence                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en PdO                                                                                                                                                                                           | Schéma de base, pour limiter le nombre de <code>threads</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence interne Théorie des threads et locks Interfaces graphiques Gestion des erreurs et exceptions Analyse | <p><b>Idée</b> : réutiliser un même <code>thread</code> pour plusieurs exécuter plusieurs tâches tour à tour.</p> <p><b>Exécuteur</b> : objet qui gère un certain ensemble de <code>threads</code></p> <ul style="list-style-type: none"> <li>• en distribuant des <b>tâches</b> sur ceux-ci, selon politique définie ;       <ul style="list-style-type: none"> <li>en évitant de créer plus de <b>threads</b> que nécessaire<sup>434</sup> ;</li> <li>et en évitant de détruire un <code>thread</code> aussitôt qu'il est libre (pour éviter d'en re-créer).</li> </ul> </li> </ul> <p><b>Tâche</b></p> <ul style="list-style-type: none"> <li>• séquence d'instructions (en pratique : une fonction) à exécuter sur un <code>thread</code></li> <li>• ne peut pas être mise en pause pour libérer le <code>thread</code> au profit d'une autre.<sup>435</sup> ↗ ceurs.</li> <li>• 434. Selon politique de l'exécuteur. Plusieurs possibles. Par exemple : nb. max. <code>threads</code> ≤ nb. cœurs.</li> <li>• 435. Le <code>thread</code>, lui, peut être mis en pause par le noyau pour libérer un processeur au profit d'un autre <code>thread</code>.</li> <li>• 436. En principe, car avec <code>ForkJoinPool</code>, notamment, certains appels de méthodes permettent la mise en pause → multi-tâche coopératif.</li> </ul> |
| Compléments en PdO                                                                                                                                                                                           | <b>Interfaces ExecutorService, Callable&lt;V&gt; et Future&lt;V&gt;</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

| APIs pour la concurrence                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en PdO                                                                                                                                                                                           | Diférents styles d'APIs pour les tâches                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence interne Théorie des threads et locks Interfaces graphiques Gestion des erreurs et exceptions Analyse | <p>Pour synchroniser et faire communiquer des tâches interdépendantes, 2 styles d'API : (dans les 2 cas, passage de <b>messages</b> plutôt que variables partagées)</p> <ul style="list-style-type: none"> <li>• <b>API bloquante</b> : appel de méthode bloquante pour attendre la fin d'une autre tâche (comme <code>join</code> pour les <code>threads</code>) et obtenir son résultat (si applicable).</li> <li>• <b>Exemple :</b></li> </ul> <pre>public class ForkJoinTask&lt;Integer&gt; {     public void adapt(Callable&lt;Integer&gt; f) {         ForkJoinTask&lt;Integer&gt; f2 = ForkJoinTask.adapt(f);         f2.fork();         System.out.println("lancement d'une sous-tâche");         f2.join();         System.out.println("appel bûquant avec récupération du résultat");     } }</pre> <p>Dans le <b>JDK</b> : <code>Thread</code>, <code>Future</code> et <code>ForkJoinTask</code> suivent ce principe.<sup>437</sup></p> <p><u>437.</u> Hors JDK, citons le principe des « fibres » dans la bibliothèque Quasar (qui implémente aussi les « acteurs » en API bloquante).</p> |
| Compléments en PdO                                                                                                                                                                                           | <b>Interfaces ExecutorService, Callable&lt;V&gt; et Future&lt;V&gt;</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |

| Interfaces Executor et Runnable                                                                                                                                                                              |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en PdO                                                                                                                                                                                           | <b>Interfaces ExecutorService, Callable&lt;V&gt; et Future&lt;V&gt;</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence interne Théorie des threads et locks Interfaces graphiques Gestion des erreurs et exceptions Analyse | <p>En Java, les exécuteurs sont les instances de l'interface <code>Executor</code> :</p> <pre>public interface Executor {     void execute(Runnable command); }</pre> <p>L'appel <code>unExecutor.execute(unRunnable)</code> exécute la méthode <code>run()</code> de <code>unRunnable</code>. Ainsi, dans ce cas, les tâches sont des instances de <code>Runnable</code>.</p> <p><b>Un exemple :</b> <code>(ExecutorService étend Executor)</code></p> <pre>ExecutorService executor = Executors.newSingleThreadExecutor(); // instantiation d'un exécuteur gérant un thread unique : // lancement d'une tâche (décrise par la lambda expression, type Inferné Runnable) executor.execute(() -&gt; System.out.println("bla, bla")); // on détruit les threads dès que tout est terminé executor.shutdown();</pre> <p>Implémentations diverses, utilisant un ou plusieurs <code>threads</code> (<b>thread pool</b>).</p> |
| Compléments en PdO                                                                                                                                                                                           | <b>Interfaces ExecutorService, Callable&lt;V&gt; et Future&lt;V&gt;</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

| APIs pour la concurrence                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en PdO                                                                                                                                                                                           | Diférents styles d'APIs pour les tâches                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence interne Théorie des threads et locks Interfaces graphiques Gestion des erreurs et exceptions Analyse | <p><b>API non bloquante</b> : une tâche qui dépend d'un résultat fourni par une autre est passée en tant que <b>fonction de rappel</b> (<code>callback</code>).<sup>438</sup> Cette dernière sera déclenchée par l'arrivée du résultat attendu (plus généralement : un événement). <b>Exemple :</b></p> <pre>CompletableFuture&lt;String&gt; completableFuture = CompletableFuture.supplyAsync(scanner::nextLine); CompletableFuture&lt;String&gt; completableFuture2 = completableFuture.thenAccept(System.out::println); completableFuture2.thenRun(() -&gt; scanner.close());</pre> <p><b>Dans le JDK Swing, CompletableFuture, Stream, Flow (Java 9)</b><sup>439</sup></p> <p><u>438.</u> Sur le principe, une fonction de première classe → en Java traditionnel un objet avec une méthode dédiée, en Java moderne, une lambda-expression.</p> <p><u>439.</u> Hors JDK : Akka (implémentation des acteurs), diverses implémentations de Reactive Streams (autres que F洛w), JavaFX (en effet, JavaFX n'est plus dans le JDK depuis Java 11), ...</p> |
| Compléments en PdO                                                                                                                                                                                           | <b>Interfaces ExecutorService, Callable&lt;V&gt; et Future&lt;V&gt;</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |

| Interfaces ExecutorService, Callable<V> et Future<V>                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en PdO                                                                                                                                                                                           | Usage (schéma général)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence interne Théorie des threads et locks Interfaces graphiques Gestion des erreurs et exceptions Analyse | <pre>public class TestCall implements Callable&lt;Integer&gt; {     private final int x;     public TestCall(int x) {         this.x = x;     }     @Override     public Integer call() {         return x;     } }</pre> <pre>public class Example {     public static void main(String[] args) {         ExecutorService executor = Executors.newTestCall(1);         Future&lt;String&gt; future1 = executor.submit(new TestCall(2));         try {             System.out.println(future1.get());         } catch (ExecutionException e) {             if (e instanceof InterruptedException) {                 Thread.currentThread().interrupt();             }             finally {                 executor.shutdown();             }         }     } }</pre> <p>Ici, l'exécuteur exécute les 2 tâches l'une après l'autre (un seul <code>thread</code> utilisé), mais en même temps que la méthode <code>main()</code> continue à s'exécuter.</p> <p>Cette dernière finit par attendre les résultats des 2 tâches pour les additionner.</p> |
| Compléments en PdO                                                                                                                                                                                           | <b>Exemple</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |

| Interfaces ExecutorService, Callable<V> et Future<V>                                                                                                                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en PdO                                                                                                                                                                                           | <b>A l'aide de la classe Executors :</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Introduction Généralités Style Objets et classes Types et polymorphisme Héritage Généricité Concurrence interne Théorie des threads et locks Interfaces graphiques Gestion des erreurs et exceptions Analyse | <ul style="list-style-type: none"> <li>= bibliothèque de fabriques statiques d'<code>ExecutorService</code>.</li> <li><b>static ExecutorService newSingleThreadExecutor ()</b> : crée un <code>ExecutorService</code> utilisant un <i>worker thread</i> unique.</li> <li><b>static ExecutorService newCachedThreadPool ()</b> : crée un exécuteur dont les <code>threads</code> du pool se créent, à la demande, sans limite, mais sont réutilisés s'ils reviennent disponibles.</li> <li><b>static ExecutorService newFixedThreadPool(int nThreads)</b> : même chose, mais avec limite fixée à <code>n</code> <code>threads</code>.</li> </ul> <p>On peut aussi utiliser directement les constructeurs de <code>ThreadPoolExecutor</code> ou de <code>ScheduledThreadPoolExecutor</code> (nombres options).</p> |
| Compléments en PdO                                                                                                                                                                                           | <b>Détails</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |

440. Choisir en rapport avec le nombre d'unités de calcul/cœurs.

441. Implémenté `ScheduledExecutorService`, permettant de programmer des tâches périodiques et/ou différées. Les futurs retournés implémentent `ScheduleFuture<T>`. Regarder la documentation.

| Les limites de ThreadPoolExecutor                                                                                                                                                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en POO<br>Aidez Degorre                                                                                                                                                                                                                     | (et des fabricues newCachedThreadPool() et newFixedThreadPool())                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| Introduction<br>Généralités<br>Style<br>Objets et classes<br>Types et polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>ressources<br>Threads et Java<br>synchronisation et locks<br>Interfaces graphiques<br>Gestion des erreurs et exceptions | <ul style="list-style-type: none"> <li>But d'un <i>thread pool</i> = réduire le nombre de <i>threads</i> → petit nombre de <i>threads</i>.</li> <li>Or les <i>threads</i> bloqués (par appelle bloquant comme <code>f.get()</code>, au sein de la tâche) ne sont pas éattribuables à une autre tâche 442. <ul style="list-style-type: none"> <li>→ moins de <i>threads</i> disponibles dans le <i>pool</i> → ralentissement.</li> </ul> </li> <li>Cas extrême : si grand nombre de tâches concurrentes avec interdépendances, il arrive que tout le <i>pool</i> soit bloqué par des tâches en attente de tâches bloquées ou non démarriées → rien ne viendra débloquer la situation.</li> </ul> <p>Cette situation s'appelle un <b>thread starvation deadlock</b> 443.</p> <p>Comment concilier <i>pool</i> de taille bornée et garantir d'absence de blocage ?</p> <p>442. Il est impossible de « sortir » une tâche déjà en exécution sur un <i>thread</i> pour le libérer.<br/> 443. Du point de vue des <i>threads</i>, c'est bien un <i>deadlock</i> : dépendance cyclique entre <i>threads</i>. Du point de vue des tâches, dépendance pas forcément cyclique, mais blocage car multi-tâche non-préemptif s'exécutant sur un nombre limité d'unités d'exécution.</p> |

| Work-stealing strategy                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en POO<br>Aidez Degorre                                                                                                                                                                                                                     | Indications et contre-indications                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| Introduction<br>Généralités<br>Style<br>Objets et classes<br>Types et polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>ressources<br>Threads et Java<br>synchronisation et locks<br>Interfaces graphiques<br>Gestion des erreurs et exceptions | <p><b>Solution :</b> stratégie de vol de travail (<b>work stealing</b>). Principe :</p> <ul style="list-style-type: none"> <li>une file d'attente de tâches par <i>thread</i> au lieu d'une commune à tout le <i>pool</i>;</li> <li>tâches générées par une autre tâche → ajoutées sur file du même <i>thread</i>;</li> <li>quand un <i>thread</i> veut du travail, il prend une tâche en priorité dans sa file, sinon il en « vole » une dans celle d'un autre <i>thread</i>;</li> <li>plus important : si la tâche attendu n'est pas disponible, <code>get()</code> (et <code>join()</code>) exécute d'abord les tâches en file au lieu de bloquer le <i>thread</i> tout de suite.</li> </ul> <p>⇒ C'est là que se met en place la coopération entre tâches.</p> |

| Work-stealing strategy                                                                                                                                                                                                                                  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en POO<br>Aidez Degorre                                                                                                                                                                                                                     | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Introduction<br>Généralités<br>Style<br>Objets et classes<br>Types et polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>ressources<br>Threads et Java<br>synchronisation et locks<br>Interfaces graphiques<br>Gestion des erreurs et exceptions | <p>Le vol de travail assure que si tous les <i>threads</i> sont bloqués (par des tâches en attente d'une autre tâche), c'est qu'il n'y a plus de tâche à démarrer.</p> <p>Cela veut dire que les tâches attendus sont déjà démarriées et non terminées. C'est donc qu'elles sont elles-mêmes en attente d'une tâche... aussi en attente.</p> <p>⇒ la seule possibilité c'est que les tâches s'attendent les unes les autres forment un (ou des) cycle(s) de dépendances.</p> <p>⇒ si pas de dépendances cycliques, thread starvation deadlock impossible.</p> |

| ForkJoinPool et ForkJoinTask                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en POO<br>Aidez Degorre                                                                                                                                                                                                                     | Implémentation de la work-stealing strategy en Java                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Introduction<br>Généralités<br>Style<br>Objets et classes<br>Types et polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>ressources<br>Threads et Java<br>synchronisation et locks<br>Interfaces graphiques<br>Gestion des erreurs et exceptions | <p>Ainsi Java propose :</p> <ul style="list-style-type: none"> <li>la classe <code>ForkJoinPool</code> : implémentation d'<code>Executor</code> utilisant cette stratégie</li> <li>la classe <code>ForkJoinTask</code> : tâches capables de générer des sous-tâches (<code>fork()</code>) et d'attendre leurs résultats pour les utiliser (<code>join()</code>).</li> </ul> <p><code>ForkJoinPool</code> est considéré suffisamment efficace pour que le <i>thread pool</i> par défaut (utilisé implicitement par plusieurs API concurrentes) soit une instance de cette classe.</p> <p>Obtenir le <i>thread pool</i> par défaut : <code>ForkJoinPool.commonPool()</code>.</p> <p>444. Sans compter qu'en Java, l'implémentation de celle-ci (<code>ForkJoinPool</code>) est plus configurable que l'implémentation de celle-ci (<code>ForkJoinPool</code>).</p> |

| ForkJoinPool et ForkJoinTask                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en POO<br>Aidez Degorre                                                                                                                                                                                                                     | Méthodes de ForkJoinTask :                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Introduction<br>Généralités<br>Style<br>Objets et classes<br>Types et polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>ressources<br>Threads et Java<br>synchronisation et locks<br>Interfaces graphiques<br>Gestion des erreurs et exceptions | <ul style="list-style-type: none"> <li><code>ForkJoinTask</code> : demande l'exécution du <i>task</i> et rend la main. Le résultat du calcul peut être récupéré plus tard en interrogeant l'objet retourné.</li> <li><code>T invoke()</code> : pareil, mais attend que soit finie et retourne le résultat.</li> <li><code>T join()</code> : attende le résultat du calcul signifié par <code>T get()</code> existe aussi car <code>ForkJoinTask</code> implemente <code>Future&lt;?&gt;</code></li> </ul> <p><code>Fork</code> et <code>invoke</code> exécutent la tâche dans le <i>pool</i> dans lequel elles sont appelées, si applicable, sinon dans le <i>pool</i> par défaut.</p> <ul style="list-style-type: none"> <li>Pour exécuter une tâche sur un <code>ForkJoinPool</code> précis, on appelle les méthodes suivantes (de la classe <code>ForkJoinPool</code>) sur le <i>pool</i> <i>pool</i> :</li> <li>&lt;<code>T invoke(ForkJoinTask&lt;?&gt; task)</code> : demande l'exécution de <i>task</i> sur <i>pool</i> et retourne le résultat dès qu'elle se termine (appel bloquant).</li> <li>&lt;<code>T Future&lt;?&gt; submit(ForkJoinTask&lt;?&gt; task)</code> : idem, mais rend la main tout de suite en retournant un futur, permettant de récupérer le résultat plus tard.</li> <li><code>void execute(ForkJoinTask&lt;?&gt; task)</code> : idem, aussi non bloquant, mais on ne récupère pas le résultat.</li> </ul> |

| ForkJoinPool et ForkJoinTask                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en POO<br>Aidez Degorre                                                                                                                                                                                                                     | Et car rien ne vaut un exemple...                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| Introduction<br>Généralités<br>Style<br>Objets et classes<br>Types et polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>ressources<br>Threads et Java<br>synchronisation et locks<br>Interfaces graphiques<br>Gestion des erreurs et exceptions | <p><b>La tâche récursive :</b></p> <pre>class Fibonacci extends RecursiveTask&lt;Integer&gt; {     final int n;     Fibonacci(int n) { this.n = n; }     @Override protected Integer compute() {         if (n &lt;= 1) return n;         Fibonacci f1 = new Fibonacci(n - 1);         Fibonacci f2 = new Fibonacci(n - 2);         return f2.compute() + f1.join();     } }</pre> <p>Et l'appel initial (dans main()) :</p> <pre>System.out.println(new ForkJoinPool().invoke(new Fibonacci(12)));</pre> |

| Une alternative : CompletetableFuture                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Compléments en POO<br>Aidez Degorre                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Introduction<br>Généralités<br>Style<br>Objets et classes<br>Types et polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>ressources<br>Threads et Java<br>synchronisation et locks<br>Interfaces graphiques<br>Gestion des erreurs et exceptions | <ul style="list-style-type: none"> <li><b>class CompletetableFuture&lt;T&gt; implements Future&lt;T&gt;, CompletionStage&lt;T&gt;</b></li> <li><b>CompletionStage&lt;T&gt; propose des méthodes pour ajouter des callbacks à exécuter lors de la complétion de la tâche.</b></li> </ul> <p>→ changement de paradigme : plus d'appels bloquants dans les tâches élémentaires, mais dépendances maintenant décrites en dehors de celles-ci.</p> <p>Le programme appelant compose ces tâches entre elles pour décrire une « recette » 447 qui il soumet au <i>thread pool</i> (et donc n'éffectue pas non plus d'appel bloquant).</p> |

## Une alternative : CompletableFuture

Compléments en PdO  
Aide/Dégorge

### Principes généraux

- ```
Rf rf = Boulot.rf();  
Future<Rf> ffg = ForkJoinTask.adapt(() -> Boulot.g(rf)).fork();  
Rf rf2 = Boulot.rf();  
Rf rf3 = Boulot.rf();  
System.out.println("Resultat: " + rf1);
```
- Devient :**
- ```
CompletableFuture<Rf> cff = CompletableFuture.supplyAsync(Boulot::f);  
CompletableFuture<Rf> cff2 = CompletableFuture.supplyAsync(() -> cff.thenApplyAsync(rf1 -> {  
    return rf2; }));  
CompletableFuture<Rf> cff3 = cff.thenApplyAsync(rf1 -> {  
    return rf3; });
```

Exemple

## Interfaces graphiques en Java

Compléments en PdO  
Aide/Dégorge

### Pour des raisons historiques, plusieurs bibliothèques sont utilisables (y compris dans le JDK) :

- AWT : existe depuis les premières versions de Java, se repose sur les composants graphiques "naïfs" du système d'exploitation (rapide, mais apparence différente entre Windows, macOS, Linux, etc....)
  - Swing : bibliothèque "officielle" de Java. Dépend peu des composants du système (donc apparence différente entre plateformes).
  - SWT (et successeur JFace) : bibliothèque du projet Eclipse. Se repose principalement sur les composants natifs (comme AWT), mais implémente tout ce que le système ne fournit pas.
  - JavaFX : alternative plus moderne, similaire à Swing dans les principes. [448](#)
- Dans ce cours : exemple de JavaFX, mais principes similaires pour les autres.  
[448](#). Intégrer un temps au JDK comme potentiel successeur de Swing (Java 8), puis finalement confié au projet OpenJFX (Java 11).

## Interfaces graphiques en Java

Avertissement

Compléments en PdO  
Aide/Dégorge

### Pour des raisons historiques, plusieurs bibliothèques sont utilisables (y compris dans le JDK) :

- Attention, c'est un sujet très vaste, même en se limitant à JavaFX.
- Ce cours ne fera qu'éclairer certains des sujets essentiels et donner quelques pointeurs pour mener à bien le projet.
- Il conviendra d'avoir une démarche active pour combler d'éventuels besoins non couverts par le cours (consultez les pages de documentation de Java<sup>449</sup>).

## Construire une GUI : mode d'emploi (1)

Compléments en PdO  
Aide/Dégorge

### Pour une fenêtre "statique" :

- 1 Conceptualisez d'abord la fenêtre de votre application, dessin à l'appui.
  - 2 Déterminez ses composants et leur hiérarchie (qui contient qui?) sous forme d'arbre.
  - 3 Programmez/écrivez [449](#) la description de la fenêtre, de ses composants et de leurs propriétés (taille, couleur, etc.) et des relations entre composants (notamment relations contentant/contenu).
- À ce stade, votre programme peut afficher votre jolie fenêtre... qui ne fera rien [450](#). Pour en faire une application utile, il faut maintenant associer des actions aux événements.

- [449](#). En fonction du contexte, ça peut être un programme Java... ou bien un fichier dans un langage descriptif tel que HTML ou XML.  
[450](#). On a en fait implémenté la partie "Vue" du patron MVC.

## Construire une GUI : mode d'emploi (2)

Gérer les événements

Compléments en PdO  
Aide/Dégorge

### Pour gérer les événements [451](#) :

- 1 On crée des **gestionnaires d'événement**, spécifiques à chaque type d'événement.
  - 2 Pour chaque type d'événement qu'on veut traiter sur un composant donné, on lui associe un gestionnaire, en le passant à une certaine méthode de ce composant. (Ceci peut se faire lors de l'initialisation du composant en question.)
  - 3 Désormais, à chaque fois que cet événement se produira, le gestionnaire sera exécuté avec pour paramètre un **déscripteur d'événement**.
- Un gestionnaire d'événement est une "fonction" [452](#) dont le paramètre est un **déscripteur d'événement** (de type souvent nommé `XXXEvent`), contenant la description des circonstances de l'événement (composant d'origine, coordonnées, bouton cliqué ...).

- [451](#). Cela correspond à la partie "Contrôleur" du patron MVC.

- [452](#). Fonction de rappel, matérialisée comme un objet contenant une méthode qui décrit la fonction ; c'est donc une fonction de première classe.

## Exécution de l'IG en JavaFX

Compléments en PdO  
Aide/Dégorge

### En JavaFX, nous avons la hiérarchie suivante :

- La branche est appelé **graphhe de scène** (`scene graph`) et est constitué de **nœuds**, instances de sous-classes de `Node`.
  - Les nœuds internes sont instances de sous-classes de `Parent`.
  - Le nœud racine de l'arbre est associé à un objet de classe `Scene`. La **scène** correspond à la totalité de l'IG destinée à effectuer une tâche donnée.
  - La scène, pour être affichée, doit être donnée à un objet de classe `Stage` [454](#) (le lieu où sera dessinée l'IG ; p. ex., sur un ordinateur de bureau : une fenêtre).
- Cette organisation permet de facilement changer le contenu entier d'une fenêtre pour passer d'une tâche à l'autre : il suffit de dire au stage d'afficher une autre scène.

- [454](#). **scène et stage** : les deux se traduisent en Français par "scène" mais ont un sens très différent. Scène désigne une subdivision temporelle (scène = chapitre d'une pièce de théâtre), alors que stage désigne un lieu (scène = les planches sur lesquelles on joue la pièce). Ainsi, pour éviter les confusion, soit je ne traduirai pas stage soit je dirai juste... une fenêtre !

## Structure d'une IG en JavaFX

Compléments en PdO  
Aide/Dégorge

### En JavaFX, nous avons la hiérarchie suivante :

- A existé tantôt comme bibliothèque séparée, tantôt comme composant du JDK (java 8 à 10). A partir de Java 11, développé au sein du projet OpenJFX.
  - Avant Java 8, JavaFX pouvait être programmé via un langage de script appelé `JavaFX Script`, celui-ci a été abandonné depuis.
  - JavaFX : bien plus qu'une bibliothèque de description d'IG.  
Par exemple, en plus des composants typiques, on peut insérer dans l'arbre des formes 3D, et appliquer au tout diverses transformations géométriques, y compris en 3D.
- Une particularité de JavaFX c'est la possibilité de décrire une IG via un langage de description appelé FXML (inspiré de HTML), et de définir le style des composants via des pages de style CSS. [453](#)

- [453](#). Ce cours n'explique pas la syntaxe de FXML et de CSS, mais seulement de la construction de l'IG via des méthodes purement Java. Cependant, vous pouvez les utiliser en TP ou en projet.

## Exécution de l'IG en JavaFX

Compléments en PdO  
Aide/Dégorge

### En JavaFX, nous avons la hiérarchie suivante :

- La branche est appelé **graphhe de scène** (`scene graph`) et est constitué de **nœuds**, instances de sous-classes de `Node`.
  - Les nœuds internes sont instances de sous-classes de `Parent`.
  - Le nœud racine de l'arbre est associé à un objet de classe `Scene`. La **scène** correspond à la totalité de l'IG destinée à effectuer une tâche donnée.
  - La scène, pour être affichée, doit être donnée à un objet de classe `Stage` [454](#) (le lieu où sera dessinée l'IG ; p. ex., sur un ordinateur de bureau : une fenêtre).
- Cette organisation permet de facilement changer le contenu entier d'une fenêtre pour passer d'une tâche à l'autre : il suffit de dire au stage d'afficher une autre scène.

- [454](#). **scène et stage** : les deux se traduisent en Français par "scène" mais ont un sens très différent. Scène désigne une subdivision temporelle (scène = chapitre d'une pièce de théâtre), alors que stage désigne un lieu (scène = les planches sur lesquelles on joue la pièce). Ainsi, pour éviter les confusion, soit je ne traduirai pas stage soit je dirai juste... une fenêtre !

|                                                                                                                                       |                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                                                      |
|---------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| Compléments en PdO<br>Affilié/Degre                                                                                                   | Introduction<br>Généralités<br>Style<br>Objets et classes<br>Interfaces polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>Interaction entre les groupes de classes<br>Prise en charge des sous-classes<br>JavaFX<br>Sémantique<br>Gestion des erreurs et exceptions | Gérer les événements<br>Compлементs en PdO<br>Affilié/Degre                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     | Gérer les événements en JavaFX<br>Complement en PdO<br>Affilié Degre |
| Pour gérer les événements,<br>les gestionnaires d'événement de JavaFX implémentent l'interface<br><code>EventHander&lt;T&gt;</code> : | <pre>public interface EventHandler&lt;T extends Event&gt; {     void handle (T e); }</pre> (où <code>T</code> peut être remplacé par le type d'événement à traiter, ex :<br><code>ActionEvent</code> , <code>KeyEvent</code> , <code>MouseEvent</code> , ...).              | <p>② on associe un gestionnaire d'événement à un composant JavaFX ainsi :</p> <pre>composant .setOnMousePressed(EventHandler&lt;? super MouseEvent&gt; gest)</pre> à partir de désormais, quand un événement du type indiqué se produit, la méthode <code>handle()</code> du gestionnaire est exécutée.                                                                                                                                                                                                                                                                                                                                         | 458.                                                                 |
| Pour gérer les événements,<br>les gestionnaires d'événement de JavaFX implémentent l'interface<br><code>EventHander&lt;T&gt;</code> : | <pre>public class GererEnregistrement implements EventHander&lt;ActionEvent&gt; {     private final Document doc;     public GererEnregistrement (Document doc) { this.doc = doc; }     public void handle (ActionEvent e) { doc.enregistre (e); } }</pre>                  | <p>Prenons l'exemple d'<code>ActionEvent</code>. Je peux par exemple créer la classe :</p> <p>Supposons maintenant que la variable <code>buttonEnregistrer</code> désigne un composant de type <code>Button</code>, alors pour que cliquer sur le bouton déclenche l'enregistrement, il suffit d'ajouter l'instruction :</p> <pre>buttonEnregistrer .setOnAction (new GererEnregistrement (documentCourant));</pre> <p>Remarque : si le objet événement, alors <code>e.getSource()</code> référence le composant où l'événement a été créé. Cette référence peut servir faire un traitement différencié en fonction de l'état du composant.</p> | 457. le contrôleur du patron MVC                                     |

|                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                             |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| EventHandler étant une interface fonctionnelle, lambda-expressions possibles :                                                                                                                                                                                              | Pour les gestionnaires courts, préférer les <u>lambda-abstractions</u> :                                                                                                                                                                                                    | Pour les gestionnaires longs, écrire un méthod et en passer une référence :                                                                                                                                                                                                 | Pour les gestionnaires longs, écrire un méthod et en passer une référence :                                                                                                                                                                                                 |
| utilisez les lambdas-expressions !                                                                                                                                                                                                                                          | <pre>composant .setOnMousePressed(e -&gt; { /* gérer l'événement e ici */ });</pre>                                                                                                                                                                                         | <pre>composant .setOnMousePressed((compteur::methodDeBoutonClic) ); // référence de méthode</pre>                                                                                                                                                                           | <pre>void methodeDeBoutonClic(MouseEvent e) { /* gérer l'événement e ici */ }</pre>                                                                                                                                                                                         |
| Dernière technique intéressante si programme plus long qu'un simple exemple : la partie où on associe composants et gestionnaires est plus succincte et claire. En plus, on peut regrouper les méthodes de gestion d'événement dans dans une même classe. <sup>457</sup>    | 457. le contrôleur du patron MVC                                                                                                                                                                                                                                            |                                                                                                                                                                                                                                                                             |                                                                                                                                                                                                                                                                             |
| Threads en JavaFX                                                                                                                                                                                                                                                           | Threads en JavaFX                                                                                                                                                                                                                                                           | Exécution des événements                                                                                                                                                                                                                                                    | Organiser une application graphique                                                                                                                                                                                                                                         |
| Introduction<br>Généralités<br>Style<br>Objets et classes<br>Interfaces polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>Interaction entre les groupes de classes<br>Prise en charge des sous-classes<br>JavaFX<br>Sémantique<br>Gestion des erreurs et exceptions | Introduction<br>Généralités<br>Style<br>Objets et classes<br>Interfaces polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>Interaction entre les groupes de classes<br>Prise en charge des sous-classes<br>JavaFX<br>Sémantique<br>Gestion des erreurs et exceptions | Introduction<br>Généralités<br>Style<br>Objets et classes<br>Interfaces polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>Interaction entre les groupes de classes<br>Prise en charge des sous-classes<br>JavaFX<br>Sémantique<br>Gestion des erreurs et exceptions | Introduction<br>Généralités<br>Style<br>Objets et classes<br>Interfaces polymorphisme<br>Héritage<br>Généricité<br>Concurrency<br>Interaction entre les groupes de classes<br>Prise en charge des sous-classes<br>JavaFX<br>Sémantique<br>Gestion des erreurs et exceptions |

|                                                                                                                                  |
|----------------------------------------------------------------------------------------------------------------------------------|
| Attention quand vous trouvez de la documentation : vérifiez qu'elle concerne bien la version installée chez vous. <sup>462</sup> |
|----------------------------------------------------------------------------------------------------------------------------------|

<sup>462</sup>. Les moteurs de recherche tentent encore trop à référence d'anciennes versions comme JavaFX 2...

<sup>458</sup>. Interactions indésirables, accès en compétition, ...cf. chapitre sur la concurrence

<sup>459</sup>. Vous pouvez, à effet, créer un thread classique ou, mieux, utiliser `javafx.concurrent.Worker`, API prévue par JavaFX, ou bien toute autre API de votre convenance.

<sup>460</sup>. Entraillements indésirables, accès en compétition, ...cf. chapitre sur la concurrence

<sup>461</sup>. Elles ne s'interrompent pas les unes les autres.

<sup>462</sup>. C'est souvent une bonne idée de séparer les deux aspects suivants :

**Modèle** : cœur du programme, partie "métier". C'est ici que sont gérées, organisées, traitées les données. On y trouve les déclarations de structures de données ainsi que les méthodes implémentant les différents algorithmes traitant sur ces structures.

**Vue** : partie qui sert à présenter l'application à l'utilisateur et sur laquelle l'utilisateur agit.

**Contrôleur** : partie du programme servant à interpréter les événements (entrées de l'utilisateur dans la vue, mais pas seulement) pour agir sur le modèle (déclencher un traitement, ...) et la vue (ouvrir un dialogue, ...).

JavaFX est particulièrement adapté à mettre en œuvre une stratégie **MVC**.

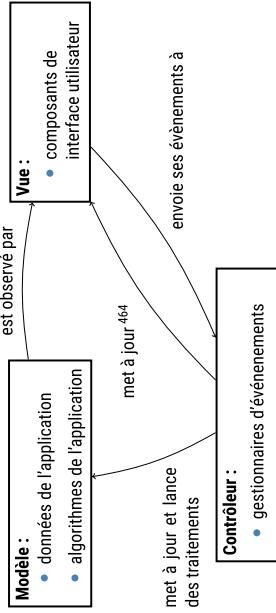
# Le patron de conception Observateur/Observable

Avec Java .util

Compléments en PdO

Autre Dégorge

# Illustration



Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

Concurrency

Interfaces

Interfaces génériques

Principes

JavaFX

Services

Gestion des erreurs et exceptions

Compléments en PdO

Autre Dégorge

Introduction

Généralités

Style

Objets et classes

Types et polymorphisme

Heritage

Généricité

## Exceptions

Factorielle corrigeé grâce aux exceptions

## Exceptions

Description du mécanisme

## Tехники де gestion des erreurs

Compléments en P00

Affiche Degrade

## Introduction

Généralités

## Style

Objets et classes

## Type et polymorphisme

Héritage

## Généricité

Concurrence

## Interfaces et graphiques

Gestion des erreurs et exceptions

## Les interfaces

Éléments communs

## La concurrence

Autres éléments

## Discussions et échanges

## Exceptions

Explication rapide

## Introduction

Généralités

## Style

Objets et classes

## Type et polymorphisme

Héritage

## Généricité

Concurrence

## Interfaces et graphiques

Gestion des erreurs et exceptions

## Le mécanisme

Éléments communs

## La construction try-with-resource

Introduction

Généralités

## Style

Objets et classes

## Type et polymorphisme

Héritage

## Généricité

Concurrence

## Interfaces et graphiques

Gestion des erreurs et exceptions

## Le bloc try/catch

Détails

## Introduction

Généralités

## Style

Objets et classes

## Type et polymorphisme

Héritage

## Généricité

Concurrence

## Interfaces et graphiques

Gestion des erreurs et exceptions

## Le constructeur

Éléments communs

## La place de la déclaration

Éléments communs

## La place de l'initialisation

Éléments communs

468.

Que répondriez-vous si on vous demandait : "Quel nombre réel vaut dix divisé par zéro ?"

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions" :

- consiste à gérer un comportement "exceptionnel" du programme, sortant du flux de contrôle "normal" ;
- sert quand il n'y a pas de valeur sensée à passer dans un **return** (la méthode est dans l'incapacité de terminer normalement<sup>468)</sup>
- → on ne veut pas redonner la main à l'appelant comme si rien d'anormal ne s'était passé,
- concerne des événements "exceptionnels" = "rares" (l'exécution de ce mécanisme est en fait coûteuse).

468. Que répondriez-vous si on vous demandait : "Quel nombre réel vaut dix divisé par zéro ?"

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- lancer (et rattraper) des **exceptions** (nous allons voir ce que c'est);
- ajouter une méthode auxiliaire pour pré-valider un appel de méthode;
- utiliser un type de retour "enrichi".

469. throw = sortie exceptionnelle; return = sortie normale

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- supposez :
- que main appelle la méthode f1 qui appelle f2 qui appelle ... qui appelle fn (→ alors)
- et que fn signale une exception exn
- si fn rattrape exn, on retrouve un fil d'exécution "normal"<sup>470</sup>, sinon, on sort de fn pile d'appel de méthodes avec main en bas de la pile, fn en haut)
- et que la méthode déclarée peut signaler<sup>471</sup> une exception.

470. On exécute le catch, le finally, puis les instructions d'après.

471. La méthode contenant try ... catch ... n'est pas nécessairement celle qui appelle directement la méthode qui fait throw (traitement non local de l'erreur).

472. Sauf construction try-with-resource, voir plus loin.

473. Signaler ou lever (raise : mot clé utilisé en OCaml) ou lancer/jeter (throw)

474. Cette clause est parfois obligatoire, parfois pas, détaillé plus loin.

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Signaler une exception (grâce à l'instruction **throw**) fait sortir de la méthode sans exécuter de **return**.<sup>469</sup>
- L'exception peut ensuite être rattrapée ou non.
- Non-rattrapée, → le programme se quitte en affichant un message d'erreur
- Rattrapée, si :
  - exécution sous la portée dynamique du **try** d'un groupe try ... catch ...
  - l'exception a le type donné entre () après le catch
  - exécution du bloc d'instruction de ce catch.
- La méthode contenant try ... catch ... n'est pas nécessairement celle qui appelle directement la méthode qui fait throw (traitement non local de l'erreur).

469. throw = sortie exceptionnelle; return = sortie normale

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attribut de catch = déclaration de variable d'un sous-type de Throwable.
- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées dans l'ordre de sous-type (les sous-types ayant les supertypes) !

470. Si on inverse les deux catch, erreur de compilation.

471. error: exception Exception2 has already been caught

472. Raison : Exception2 est un cas particulier de Exception1, donc déjà traité dans le premier catch. Le deuxième catch est alors inutile.<sup>475</sup>

473. Variante ("multi-catch") : catch (Exception1 | Exception2 e){ ... }

474. Or le compilateur considère que si on écrit du code inutile, c'est involontaire et donc une erreur.

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Paramètre de catch = déclaration de variable d'un sous-type de Throwable.
- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

475. Java ≥ 9, on peut passer comme argument de try une variable déjà déclarée<sup>476</sup> à la place de la déclaration. On peut donc facilement séparer initialisation et usage.

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

476. si elle est finale ou effectivement finale.

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

477. Java ≥ 9, on peut passer comme argument de try une variable déjà déclarée<sup>476</sup> à la place de la déclaration. On peut donc facilement séparer initialisation et usage.

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

478. Variante ("multi-catch") : catch (Exception1 | Exception2 e){ ... }

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

479. Variante : FactorielNegativeException extends Exception {

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

480. Variante : FactorielNegativeException extends Exception {

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

481. Variante : FactorielNegativeException extends Exception {

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

482. Variante : FactorielNegativeException extends Exception {

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

483. Variante : FactorielNegativeException extends Exception {

## Compléments en P00

Affiche Degrade

## Le mécanisme des "exceptions"

- Attention : en cas de plusieurs clauses catch, exceptions doivent être traitées

484. Variante : FactorielNegativeException extends Exception {

## Classes d'exceptions

Description des classes proposées (1)

### Classes d'exceptions

Compléments en P00

Audrey Degorre

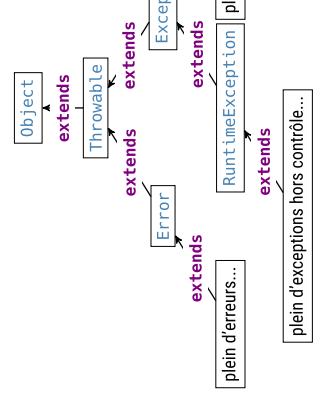
- Objet exception = objet passé en paramètre de **throw**, récupérable par **catch**.
- Instance (indirecte) de la classe **Throwable**.
- Contient de l'information utile pour récupérer l'erreur (bloc **catch**) ou bien déboguer un crash, notamment :
  - sa classe, le fait d'appartenir à une sous-classe d'exception ou une autre est déjà une information très utile.
  - message d'erreur expliquant les circonstances de l'erreur → `String getMessage()`
  - cause, dans le cas où l'exception est elle-même causée par une autre exception → `Throwable.getcause()`
  - trace de la pile d'appels, qui donne la liste des appels imbriqués successifs (numéro de ligne et méthode) qui ont abouti à ce signallement d'exception → `StackTraceElement[] getStackTrace()`
- Attention :** génération de la trace → coûteuse en temps et en mémoire.
  - Une raison pour réservrer le mécanisme des exceptions aux cas exceptionnels.



### Hierarchie thématique

Compléments en P00

Audrey Degorre






### Classe Throwable

Compléments en P00

Audrey Degorre



### Rôle = marqueur syntaxique pour throw, throws et catch → c'est le type de tout ce qui peut être lancé et rattrapé.

### Apès **throw** l'expression doit être de (super-type) **Throwable**.

### Le paramètre déclaré dans un **catch** a pour type un sous-type de **Throwable**.

### Classe Error :

### Indique les erreurs tellement graves qu'il n'y a pas de façon utile de les rattraper.

### On a le droit de les passer en argument d'un **catch**... mais ce n'est pas conseillé!

### Ex : dépassement de la capacité de la pile d'exécution (StackOverflowError).

### Classe Exception :

### Indique les erreurs tellement graves qu'il n'y a pas de façon utile de les rattraper.

### On a le droit de les passer en argument d'un **catch**... mais ce n'est pas conseillé!

### Ex : Erreurs possiblement récupérables.

### Elles ont vocation à être passées en argument d'une clause **catch**.

### Exemples :

### opération bloquante interrompu (InterruptedException).

### Exemples :

### opération bloquante interrompu (InterruptedException).

### Remarque sur la non-localité

### Quand on exécute, la JVM crashe et affiche l'exception non rattrapée (instance de E1), ainsi que toutes ses causes successives.

### Exception in thread "main" exceptions.E1: exceptions.E2: exceptions.E3 at exceptions.E3.main(E3.java:25)

### Caused by: exceptions.E2.main(E2.java:42)

### at exceptions.E2.main(E2.java:33)

### at exceptions.E2.main(args[0]) { f(); }

### ... 1 more

### Caused by: exceptions.E3.main(E3.java:38)

### at exceptions.E3.main(args[0])

### ... 2 more

### Discussion

### Deux cas :

### Les exceptions sous contrôle doivent toujours être déclarées (**throws**).

### Lesquelles ? toutes les sous-classes de **Exception** sauf celles de **RuntimeException**.

### Raison : les concepteurs de Java ont pensé souhaitable 482 d'inciter fortement à récupérer toute erreur récupérable.

### Cette clause est obligatoire pour les exceptions dites "sous contrôle" 480.

### Conséquence : si **FileNotFoundException** est sous-controle et que f() (ci-dessus) est appellée dans une autre méthode g() qui ne la rattrape pas, alors g() doit elle-même avoir **throws FileNotFoundException** et ainsi de suite.

### Du coup un programme ne peut pas crasher sur une exception sous contrôle, sauf si elle est déclarée dans la signature de `main()`. 481

### 480. voir page suivante

### 481. En fait si... on peut tricher (ce n'est pas évident), mais c'est déconseillé !

### Remarque sur la non-localité

### Non-localité = point fort des exceptions : en effet l'erreur n'est mentionnée que là où elle se produit et là où elle est traitée. 483

### Mais la non-localité peut être contradictoire avec l'encapsulation.

### → si un composant A, B doit "masquer" les exceptions de A. En effet, si C utilise B mais pas A, C ne devrait pas avoir affaire à A. 484

### Bonne conduite : traiter toutes les erreurs de A dans B. À défaut, traduire les exceptions de A en exceptions de B (en utilisant le chainage) :

```
class A { public void f() { throw new AException(); } }
```

```
private A a; public A a() { try { a.f(); } catch (AException e) { throw new BException(e); } }
```

483. Pas tout à fait vrai avec les exceptions sous contrôle.

484. Les exceptions sous contrôle rendent le problème particulièrement visible vu qu'elles sont affichées dans la signature des méthodes, mais il existe aussi avec les exceptions hors contrôle.

Discussion

### Exceptions sous contrôle vs. hors contrôle

Compléments en P00

Audrey Degorre



### La clause throws

Compléments en P00

Audrey Degorre



### La déclaration d'une méthode, la clause **throws** signale qu'une exception non rattrapée peut être lancée lors de son exécution.

```
public static void f() throws IOException { // Code pouvant générer une exception de type IOException. }
```

### Les exceptions sous contrôle et que f() (ci-dessus) est appellée dans une autre méthode g() qui ne la rattrape pas, alors g() doit elle-même avoir **throws IOException** et ainsi de suite.

### Le RuntineException peut se produire, par exemple, dès qu'on utilise le "appel de méthode, autant dire tout le temps! Si elles étaient sous contrôle, presque toutes les méthodes auraient **throws NullPointerException**!

### 482. Ce point est très controversé. Aucun langage notable, conçu après Java, n'a gardé ce mécanisme.

|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <h3>Enrichir le type de retour</h3> <p><b>Pré-valider les appels de méthode</b></p> <p>Compléments en PdO<br/>Aidez Degorre</p> <p><b>Description de la technique</b></p> <p><b>Vous avez déjà vu cette technique mise à l'œuvre dans l'API Java. Exemples :</b></p> <ul style="list-style-type: none"> <li>avec les scanners : avant de faire <code>s.c.nextInt()</code>, il faut faire <code>s.c.hasNextInt()</code> pour être sûr que le prochain mot lu est interprétable comme entier</li> <li>avec les itérateurs : avant l'appel <code>it.hasNext()</code>, on vérifie <code>it.hasNext()</code>.</li> </ul> <p><b>En résumé :</b> une méthode dont le bon fonctionnement est soumis à une certaine précondition. En général, les deux méthodes ont des noms en rapport :</p> <ul style="list-style-type: none"> <li><b>void doSomething() boolean canDoSomething()</b></li> <li><b>Foo getFoo() hasFoo()</b></li> </ul> <p><b>Le même test doit malgré tout être laissé au début de doSomething() :</b></p> <pre>if (!canDoSomething()) throw new IllegalStateException(); // ou IllegalArgumentExeception → si on oublie de tester canDoSomething au pire, crash, au lieu de résultat absurde)</pre>                                                                                                                                                                                                                                       | <h3>Pré-valider les appels de méthode</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p><b>Description de la technique</b></p> <p>Pour la factorielle, on peut créer une méthode de validation du paramètre, mais le plus simple c'est encore de penser à tester <math>x \geq 0</math> avant de l'appeler.</p> <p>La méthode factorielle s'écritra elle-même avec ce test et une exception hors contrôle :</p> <pre>static int fact(int x) { // pas de throws     if (x &lt; 0)         throw new IllegalArgumentException(); // hors contrôle     else if (x == 0)         return 1;     else         return x * fact(x - 1); }</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <h3>Enrichir le type de retour</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p><b>Description de la technique</b></p> <p>Toujours avec la factorielle, avec le type <code>Optional</code> :</p> <pre>static Optional&lt;Integer&gt; fact(int x) { // pas de throws     if (x &lt; 0) return Optional.empty();     else if (x == 0) return Optional.of(1);     else return Optional.of(x * fact(x - 1)); }</pre> <p>Comme cette méthode ne retourne pas un <code>int</code> (ou <code>Integer</code>), on n'est pas tenté d'utiliser la valeur de retour directement.</p> <p>Pour utiliser la valeur de retour :</p> <pre>Optional&lt;Integer&gt; result = fact(1); if (result.isPresent()) System.out.println(result.get()); else System.out.println("unapasdefactoriellement"); → Autre possibilité : result.ifPresent(f → System.out.println(f));</pre> <p>485. opérateur <code>-&gt;</code> introduit dans Java 8 sera à dénoter une valeur fonctionnelle. Hors programme !</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                              | <h3>Enrichir le type de retour</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p><b>Description de la technique</b></p> <p>Toujours avec la factorielle, avec le type <code>Optional</code> :</p> <pre>public class Mail {     /* attributs, constructeurs, etc. */     public static enum SendOutcome { OK, AUTH_ERROR, NO_NETWORK, PROTOCOL_ERROR }     /* send() : envoie le mail. Sans les erreurs, void suffisait...        ... mais évidemment des erreurs sont possibles.        Différents codes sont prévus dans l'enum SendOutcome */     public SendOutcome send() {         /* essaie d'enoyer le mail, mais interrompt le traitement            avec "return error.TRUC;" dès qu'il y a un souci            */         /* return SendOutcome.OK;     }</pre> <p>486. En effet : quel que soit le mécanisme utilisé, un traitement "non local" signifie qu'on doit remonter un nombre arbitraire de niveaux dans la pile d'appels.</p> <p>487. À condition de pas oublier de toutes les traiter. A cet effet, pensez à documenter les méthodes qui levrent de telles exceptions → mortel à <code>throws</code> de la JavaDoc.</p>                                                                                                                                                                                                                                                                                                                                                                                                   |
| <h3>Enrichir le type de retour</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p><b>Description de la technique</b></p> <p>Toujours avec la factorielle, avec le type <code>Optional</code> :</p> <pre>static Optional&lt;Integer&gt; fact(int x) { // pas de throws     if (x &lt; 0) return Optional.empty();     else if (x == 0) return Optional.of(1);     else return Optional.of(x * fact(x - 1)); }</pre> <p>Comme cette méthode ne retourne pas un <code>int</code> (ou <code>Integer</code>), on n'est pas tenté d'utiliser la valeur de retour directement.</p> <p>Pour utiliser la valeur de retour :</p> <pre>Optional&lt;Integer&gt; result = fact(1); if (result.isPresent()) System.out.println(result.get()); else System.out.println("unapasdefactoriellement"); → Autre possibilité : result.ifPresent(f → System.out.println(f));</pre> <p>485. opérateur <code>-&gt;</code> introduit dans Java 8 sera à dénoter une valeur fonctionnelle. Hors programme !</p>                                                                                                                                                                                                                                                                                                                                                                                                                                                              | <h3>Enrichir le type de retour</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p><b>Description de la technique</b></p> <p>Toujours avec la factorielle, avec le type <code>Optional</code> :</p> <pre>public class Mail {     /* attributs, constructeurs, etc. */     public static enum SendOutcome { OK, AUTH_ERROR, NO_NETWORK, PROTOCOL_ERROR }     /* send() : envoie le mail. Sans les erreurs, void suffisait...        ... mais évidemment des erreurs sont possibles.        Différents codes sont prévus dans l'enum SendOutcome */     public SendOutcome send() {         /* essaie d'enoyer le mail, mais interrompt le traitement            avec "return error.TRUC;" dès qu'il y a un souci            */         /* return SendOutcome.OK;     }</pre> <p>486. En effet : quel que soit le mécanisme utilisé, un traitement "non local" signifie qu'on doit remonter un nombre arbitraire de niveaux dans la pile d'appels.</p> <p>487. À condition de pas oublier de toutes les traiter. A cet effet, pensez à documenter les méthodes qui levrent de telles exceptions → mortel à <code>throws</code> de la JavaDoc.</p>                                                                                                                                                                                                                                                                                                                                                                                                   |
| <h3>Points forts et limites de chaque approche</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p><b>Description de la technique</b></p> <p>Exceptions (2)</p> <ul style="list-style-type: none"> <li>Exceptions sous contrôle :             <ul style="list-style-type: none"> <li>"Pollution syntaxique" : obligation d'ajouter au choix des <code>try</code> dans toute méthode ou une telle exception peut se produire.</li> <li>Sinon sait traiter l'exception localement, pas de problème.</li> <li>Simplement se retrouve à ajouter une clause <code>try</code> à l'appelant (qui vient s'ajouter aux autres s'il y en avait déjà...), puis à l'appelant de l'appelant, puis...</li> <li>Incitation à faire des <code>try catch</code> juste pour éviter d'ajouter un <code>throws</code> : nuisible si on fait faire le travail dans le traiteur. P. ex. :</li> </ul> </li> </ul> <pre>void [] throws Ext { throw new Ext(); } try (f1(), ...) catch (Ext e) { /* rien */ ← au fait, pas bien ! ← */ }</pre> <p>→ la pratique moderne semble éviter de plus en plus l'utilisation de ce mécanisme. Cependant l'API Java l'utilise beaucoup et il faut donc savoir comment faire avec.</p> <p>488. Mais pour un traitement local, à quoi bon utiliser les exceptions ?</p> <p>489. Modification de signature des appelants qui finalement revient à peu près à changer les types de retour par des types enrichis. → critique courante.</p> | <h3>Points forts et limites de chaque approche</h3> <p>Compléments en PdO<br/>Aidez Degorre</p> <p><b>Description de la technique</b></p> <p>Exceptions (1)</p> <ul style="list-style-type: none"> <li>Exceptions :             <ul style="list-style-type: none"> <li>programmer un "type somme" à la main</li> <li>s'il n'y a qu'un seul code d'erreur (ou bien si on ne veut pas distinguer les différentes erreurs), utiliser la classe <code>Optional&lt;T&gt;</code> (Java 8).</li> <li>si la méthode qui produit l'erreur devait être <code>void</code>, retourner à la place un <code>boolean</code> (sin on ne souhaite pas distinguer les erreurs) ou mieux, une valeur de type énuméré (chaque constante correspond à un code d'erreur).</li> </ul> </li> </ul> <p>Il s'agit d'une amélioration de la technique du "code d'erreur" : au lieu de réserver une valeur dans le type, on prend un type somme, "plus large"</p> <ul style="list-style-type: none"> <li>contenant, sans ambiguïté, les vraies valeurs de retour et les codes d'erreur,</li> <li>et tel qu'on ne puisse pas utiliser la valeur de retour directement sans passer par un getter "fait pour ça".</li> </ul> <p>Possibilités :</p> <ul style="list-style-type: none"> <li>Généralités</li> <li>Style</li> <li>Objets et classes</li> <li>Types et polymorphisme</li> <li>Héritage</li> <li>Concurrente</li> <li>Interfaces gérées</li> <li>Gestion des erreurs et exceptions</li> <li>Autres options</li> <li>Déroulé et stratégies</li> </ul> <p>Discussion</p> |

## Points forts et limites de chaque approche

Synthèse des critères

Pas toujours de choix unique et idéal quant à la stratégie de gestion d'une erreur. Les critères suivants, avec leurs exigences contradictoires, peuvent être considérés :

- localité du traitement de l'erreur (local, non local ou pas de traitement);
- coût de la vérification de la validité de l'opération avant de l'effectuer;
- sûreté, c.-à-d. obligation de traiter le cas d'erreur (contre-partie : pollution syntaxique);
- fréquence de l'erreur (totalelement évitable en corrigeant le programme, rare, fréquente, ...);
- qualité de l'encapsulation requise.

## Discussion