

Algorithmique et programmation par objets

Inf F3

Licence 2 MIASHS

Université Grenoble Alpes

Jerome.David@univ-grenoble-alpes.fr

2022-20233

<http://miashs-www.u-ga.fr/~davidjer/inff3/>

Organisation

- 13 séances de CM – jeudi de 10h15 à 11h45
- 13 séances de TD
 - 1^{er} groupe : vendredi 13h30 à 15h00
 - 2^{ème} groupe : vendredi 15h15 à 16h45
 - 3^{ème} groupe : mardi 13h30 à 15h00 (sauf 1^{ère} semaine vendredi 10h15-11h45)
- 13 séances de TP
 - 3 groupes : lundi matin ou mercredi matin
 - commencent une semaine après cours et TD
- Evaluation (sous réserve de modifs à cause du covid-19...)
 - QCM : régulièrement au début des cours
 - Seuls les 6 ou 7 meilleurs seront comptés
 - **Apporter un correcteur (blanco)**
 - TP : un TP noté à la fin
 - Exam : sur table, 2 heures

Cours 1 – Rappels et introduction

- Rappels
 - Notions de programme et langage de programmation
 - Les paradigmes de programmation
- Introduction des concepts de la programmation par objets
- Stratégies de gestion de la mémoire

Programme informatique

- Qu'est ce qu'un programme ?
 - « Un programme informatique est une séquence d'instructions qui spécifie étape par étape les opérations à effectuer pour obtenir un résultat » (wikipedia)
 - « Un programme est [...] un algorithme exprimé dans un langage de programmation » (wikipedia)

Langage de programmation

- Qu'est ce qu'un langage de programmation ?
 - Permet de décrire les structures des données manipulées et comment elles sont manipulées
 - C'est fait d'un alphabet, un vocabulaire, des règles de grammaire, et des significations
 - Chaque langage reflète un paradigme de programmation
 - Un style fondamental de programmation (wikipedia)
 - Une façon de construire la structure et les éléments des programmes (wikipedia)

Les paradigmes de programmation

- Impérative
 - Liste ordonnée (séquence) d'instructions permettant de modifier l'état de la mémoire
 - Pour trouver l'élément maximum d'un ensemble, je parcours cet ensemble (élément par élément) et je le compare à l'élément maximum depuis le début du parcours. Si cet élément est plus grand alors il devient le maximum.
- Déclarative
 - Exprime la logique du calcul sans utiliser de structure de contrôle
 - Élément maxi d'un ensemble : $\text{Max}(S) = \{x \text{ in } S \mid \text{forall } y \text{ in } S, x > y\}$
 - Fonctionnelle : basée sur l'évaluation de fonctions (sans effets de bord)
 - Logique : faits + règles de dérivation
- Orienté Objet : c'est ce qu'on va étudier
 - Objets ayant des comportements et interagissants entre eux
- JAVA est un langage orienté **objet impératif**

Programmation impérative

- Paradigme le plus proche du fonctionnement réel de la machine
- Principe : exécuter séquentiellement une suite d'instructions
 - Une séquence d'instructions est aussi appelée bloc d'instructions
- Les types principaux d'instructions :
 - **L'assignation**
 - Permet de changer l'état (la valeur) d'une variable, comme par exemple lui associer le résultat d'une opération en vue de l'utiliser plus tard
 - **Le branchement conditionnel**
 - Permet de réaliser un bloc d'instructions que si une condition donnée est satisfaite
 - **Le branchement inconditionnel**
 - Permet de d'exécuter un bloc d'instructions qui est à un autre endroit du programme
 - Sauts (goto) ou appels de procédures, fonctions, sous-programmes
 - **Les boucles**
 - Permet de répéter un bloc d'instructions un certain nombre de fois ou jusqu'à ce qu'une condition soit réalisée

Découpage en fonctions

- Consiste à découper un programme en plusieurs parties
 - Appelées fonctions, procédures, méthodes (en OO)
- Pourquoi ?
 - Pour faciliter la lecture et compréhension
 - Pour permettre de réutiliser le même bout de programme à plusieurs endroits (factorisation)

Phases de création d'un programme

- (1) Analyse / conception
- (2) Codage
- (3) Transformation du code source
 Compilation ou/et Interprétation
- (4) Test / Validation

L'analyse & conception

- Permet de définir :
 - ce que va faire le programme (quel est le problème ?)
 - et comment il va le faire (comment résoudre le problème)
- Les éléments essentiels de l'analyse
 - Identifier les données d'entrée
 - Les données que le programme va traiter
 - Concevoir la méthode employée pour résoudre le problème
 - L'algorithme
 - Spécifier le résultat
 - Les données en sorties du programme

Le codage

- Le codage consiste à transformer l'algorithme spécifié lors de l'analyse en un programme
- On choisit un langage de programmation
 - En fonction...
 - de l'utilisation (web, application classique)
 - de l'architecture (PC, ARM, autre microcontrôleur)
 - des fonctionnalités incluses (fonctions stats, librairie web, etc)
 - de la nature du problème traité et de sa facilité à être exprimé par les différents paradigmes de programmation
 - déclarative, impérative, orientée objet, etc.

La transformation du code source

- Le code source n'est pas compris tel quel par la machine
 - Il est généralement écrit dans un langage de haut-niveau (conçu pour l'humain)
- Il faut le transformer en code machine
 - La compilation : le code source est « compilé » en un « exécutable » (C, C++, Java, etc.)
 - Un exécutable n'est pas forcément compris tel quel par une machine physique, il faut parfois utiliser une machine virtuelle (comme en Java par exemple)
 - L'interprétation : le code source est « compilé » au fur et à mesure pendant la phase d'exécution (Python, Perl, R, etc.)

Test et validation

- Etape essentielle du développement logiciel qui consiste à vérifier (expérimentalement) que le programme fait exactement ce que l'on attend
- Il existe différents type de tests
 - Tests unitaires : on teste chaque portion du programme
 - Test d'intégration: on teste les portions mises ensemble
 - Tests de validation fonctionnelle : on teste que le programme fait bien ce que l'on voulait qu'il fasse (i.e. qu'il réponde à la demande du client)

Objectifs du cours

- Ce cours a comme objectifs :
 - La programmation par objets
 - Concepts de base : objets, classe, héritage
 - Les structures de données à accès direct-indicé
 - Tableaux
 - Chaîne de caractères
 - Algorithmique
 - Tri, recherche dans tableaux triés, etc.
 - Exceptions
 - Accès fichiers

Qu'est ce qu'un objet ?

- C'est une chose... une sorte de variable... une abstraction...
 - Qui a un état
 - ce sont ses caractéristiques (ses données)
 - Sur laquelle on peut effectuer des opérations
 - ce sont ses comportements
- La programmation objet consiste à résoudre son problème en représentant les composants du problème et sa solution par des objets

Question ? Quels sont les objets ?

The screenshot shows a music player window titled "Led Zeppelin - Tangerine". The interface includes a menu bar (Music, Playlist, Tools, Extras, Help), a sidebar with navigation options (Search, Library, Files, Playlists, Internet, Devices, Song info, Artist info), and a main playlist area. The playlist is titled "Playlist 17" and "My favorite songs". The playlist table has columns for Track, Title, Artist, Album, Length, and source. The current track is "Tangerine" by Led Zeppelin, with a duration of 2:58. The status bar at the bottom indicates "58 tracks - [3:00:26]" and "2 : 40".

Track	Title	Artist	Album	Length	source
3	It's All About (with Aqeel)	Blundetto	Warm My Soul	5:26	
4	Crowded places (with Akale Horns)	Blundetto	Warm My Soul	5:11	
5	Warm My Soul (with Courtney John)	Blundetto	Warm My Soul	4:03	
6	I'll Be Home Later (with Akale Horns)	Blundetto	Warm My Soul	4:17	
7	Final Good Bye (with Tommy Guerrero)	Blundetto	Warm My Soul	3:50	
8	Treat Me Like That (with Courtney John)	Blundetto	Warm My Soul	3:05	
9	Walk Away Now (with Jahdan Blakkamoore)	Blundetto	Warm My Soul	4:10	
10	Since You've Been Gone	Blundetto	Warm My Soul	4:56	
11	Hercules Dub	Blundetto	Warm My Soul	5:38	
2	Get A Move On	Mr. Scruff	Electro Swing	3:24	
	Someday	Two Door Cinema Club	Beacon		
	What You Know	Two Door Cinema Club	Tourist History		
	I'm Gonna Be (500 Miles)	The Proclaimers	Sunshine on Leith		
	Hubrist - Resonance Disaster [free DL link in desc...	hubrist		2:51	
	Hubrist - Kurzweil's Ghost [free DL link in descript...	hubrist		2:23	
	Seize the Day	Wax Tailor	Paris		
	<i>Tangerine</i>	<i>Led Zeppelin</i>	<i>Boxed Set</i>	2:58	
	Stairway to Heaven	Led Zeppelin	[Led Zeppelin IV]		
1	Brick By Brick	Arctic Monkeys	Brick By Brick	2:59	
1	Don't Sit Down 'Cause I've Moved Your Chair	Arctic Monkeys	Don't Sit Down 'Cause I've Move...	3:03	
1	Brianstorm	Arctic Monkeys	Favourite Worst Nightmare	2:50	
2	Teddy Pickers	Arctic Monkeys	Favourite Worst Nightmare	2:43	
3	D Is For Dangerous	Arctic Monkeys	Favourite Worst Nightmare	2:16	
4	Balaclava	Arctic Monkeys	Favourite Worst Nightmare	2:49	
5	Fluorescent Adolescent	Arctic Monkeys	Favourite Worst Nightmare	2:57	
6	Only Ones Who Know	Arctic Monkeys	Favourite Worst Nightmare	3:02	
7	Do Me A Favour	Arctic Monkeys	Favourite Worst Nightmare	3:27	
8	This House Is A Circus	Arctic Monkeys	Favourite Worst Nightmare	3:00	

Les principes de la pensée objet

- Toute chose est (devrait être) objet
 - Ce n'est pas toujours le cas comme avec Java
- Un programme est constitué d'un ensemble d'objets qui communiquent (se disent quoi faire) en s'envoyant des messages
- Chaque objet a son propre espace de mémoire composé d'autres objets
- Chaque objet est d'un type précis
- Tous les objets d'un type particulier peuvent recevoir le même message

Le concept de classe

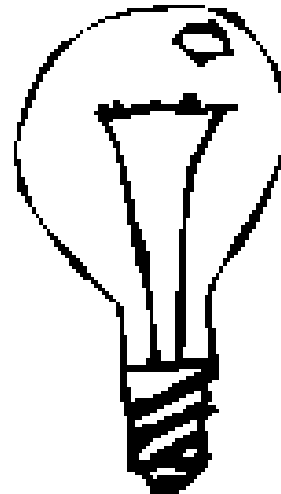
- Même si tout objet est unique, il appartient à une (ou plusieurs) classe(s) d'objets qui ont en commun
 - Des caractéristiques (du moins des types de...)
 - Des comportements
- Cette notion de CLASSE est FONDAMENTALE
- La programmation objet consiste à :
 - Créer des nouveaux types de données qu'on appelle des classes
 - Et à les utiliser : créer des objets d'un type (d'une classe) donné puis leur envoyer des messages, etc.

L'interface d'un objet

- Une fois une classe définie, on peut
 - créer autant d'objet de cette classe que l'on veut
 - On parle alors de créer des **instances** d'une classe, ou d'instancier une classe
 - les manipuler, c.-à-d. leur envoyer des messages
 - On parle alors d'appeler des méthodes de l'objet
- Chaque objet ne peut être manipulé que via son interface
 - Ce sont l'ensemble des messages que l'on peut envoyer à un objet
 - C'est la classe (son type) qui détermine son interface

Exemple

- Voici l'interface d'une ampoule électrique



- Ce sont les messages que l'on peut envoyer sur notre ampoule
 - La façon dont les messages sont traités (leur code) ainsi que les données cachées c'est ce qu'on appelle l'**implémentation**

Exemple (suite)

- Comment faire en JAVA ?

```
public class MonPremierProg {
```

1

```
public static void main(String[] args) {
```

2

On déclare une variable appelée *amp*

```
Ampoule amp = new Ampoule();  
amp.allumer();
```

On instancie un objet de type Ampoule

3

On associe l'objet créé à la variable *amp*

4

On appelle la méthode *allumer()* sur l'objet référencé par la variable *amp*

```
}  
}
```

L'encapsulation (vers la visibibilité)

- L'utilisateur d'une classe n'a a priori pas besoin
 - de savoir comment est faite l'implémentation
 - d'accéder ou de modifier certaines données d'un objet

Il ne peut accéder seulement à l'interface de l'objet
- Pour cela, il existe des « contrôles d'accès » sur les classes
 - Cela permet de réduire les bugs
 - Cela facilite la tâche du programmeur utilisateur
 - Cela permet de changer l'implémentation plus facilement

La réutilisation

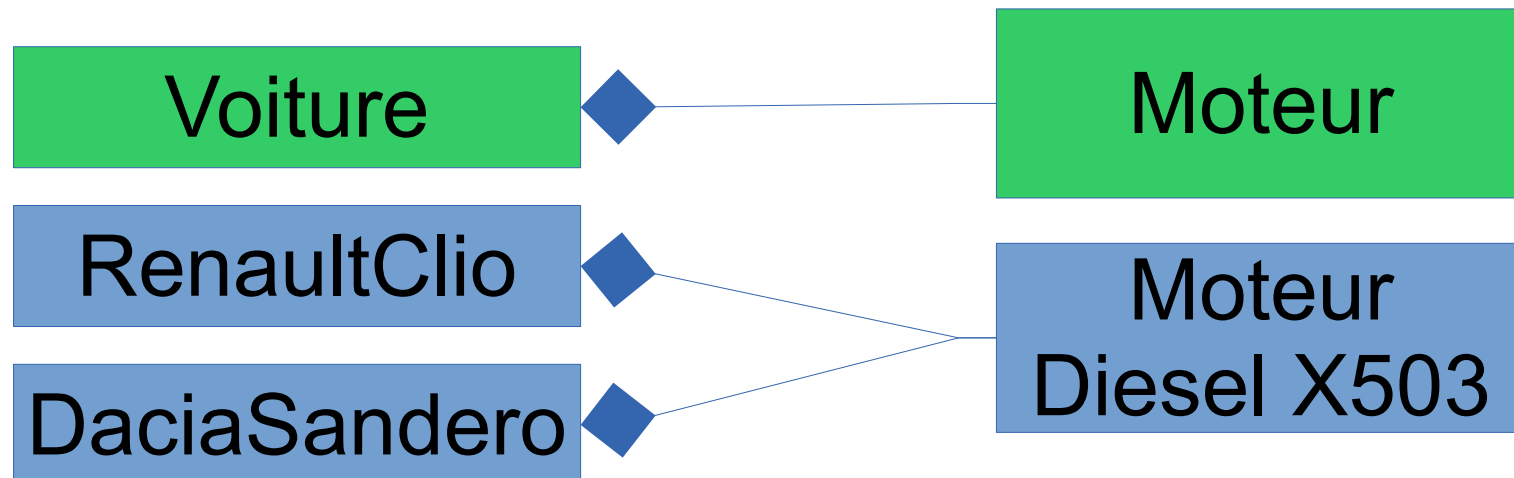
- La réutilisation de code est l'un des grands avantages des langages objets
- On distingue :
 - La réutilisation de l'implémentation
 - Via la **composition** ou plus généralement l'**agrégation**
 - La réutilisation de l'interface
 - Via l'**héritage**

Ces 2 notions sont fondamentales dans
l'approche objet

La réutilisation de l'implémentation

(vers la composition et l'agrégation)

- Le plus simple moyen de réutiliser une classe est d'utiliser un objet de cette classe dans un autre objet (→ créer un objet membre)
 - EXEMPLE
 - Une voiture est composé d'un moteur
 - Un type de moteur peut être réutilisé sur plusieurs types de voitures

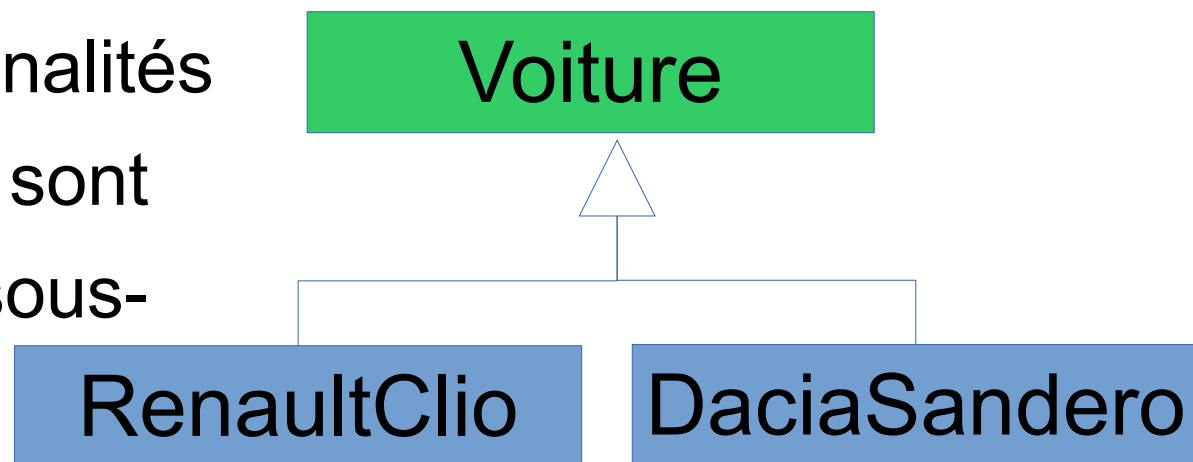


La réutilisation de l'interface

(vers l'héritage)

- Il arrive que l'on veuille réutiliser l'interface (fonctionnalités) d'une classe dans une autre
 - Il serait dommage de faire du copier-coller de la première et de la modifier
- Le mécanisme appelé **héritage** permet de faire cela

- Toutes les fonctionnalités de la super-classe sont héritées dans les sous-classes

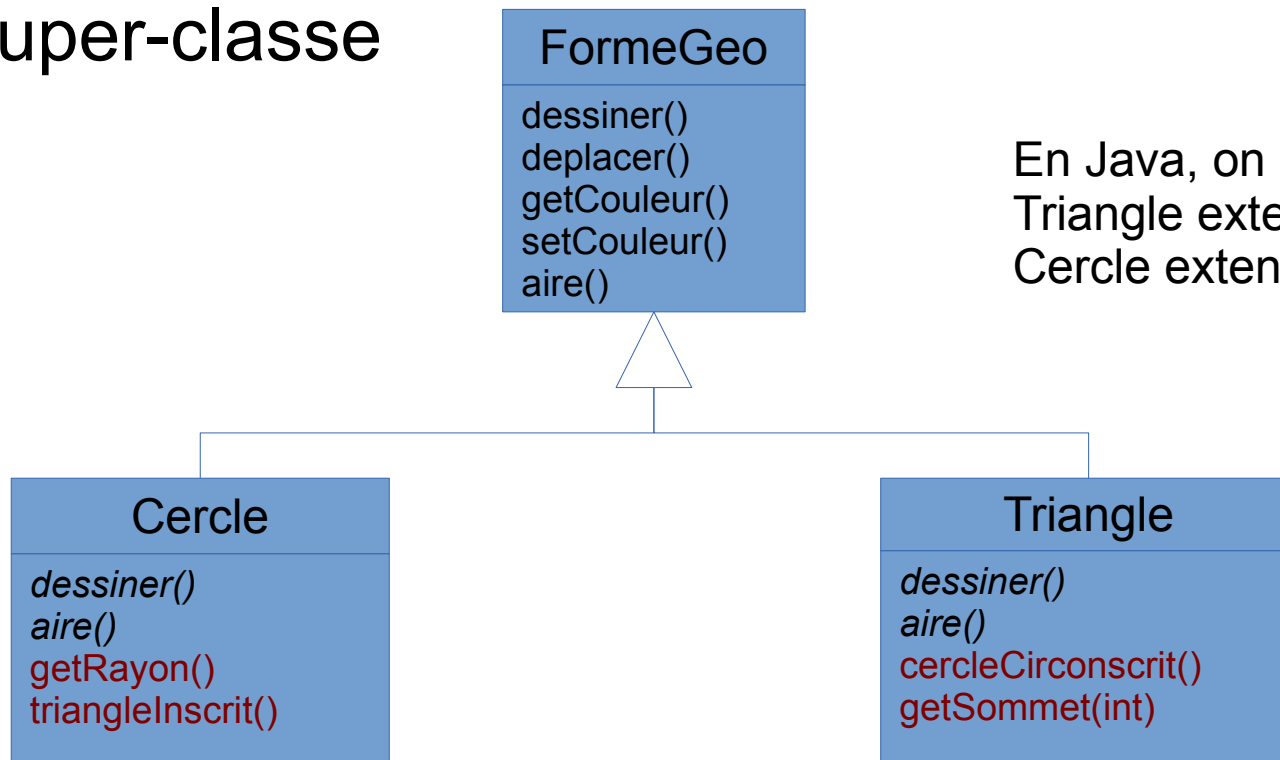


Si l'on modifie une fonctionnalité de la super classe alors elle sera modifiée dans les sous-classes

La réutilisation de l'interface

(vers l'héritage et le polymorphisme)

- Dans une sous-classe, on peut :
 - **Ajouter** des fonctionnalités (comportements)
 - **Redéfinir** des fonctionnalités par rapport à la super-classe



En Java, on dit :
Triangle extends FormeGeo
Cercle extends FormeGeo

Le polymorphisme

- C'est le principe par lequel on peut réutiliser un programme sur n'importe quel sous-type d'un type donné

– Exemple :

```
void dessineEtAfficheAire(FormeGeo f) {  
    f.dessine();  
    System.out.println("aire"+f.aire());  
}
```

Ce programme est générique : il fonctionnera pour n'importe quel sous-type de forme même si j'en crée un autre plus tard

Création et cycle de vie des objets

- A sa création un objet prend de la place en mémoire
 - On ne sait pas à priori quelle place va prendre un objet
 - La chaîne de caractères « Programmation Objet » prend plus de place que « Java »
- La mémoire n'est pas infinie
 - Il faut libérer la mémoire lorsqu'un objet n'est plus utilisé.
 - Comment savoir qu'un objet n'est plus utilisé ?
- Les langages de programmation utilisent différentes stratégies pour allouer et libérer la mémoire

L'allocation mémoire 1/3

- Il existe 3 stratégies d'allocation de mémoire
 - Allocation statique, dynamique sur la pile, dynamique sur le tas
- **L'allocation statique**
 - Se fait avant l'exécution (à la compilation)
 - Au lancement du programme, le système réserve tout l'espace dont le programme aura besoin
 - Il n'y a pas d'allocation de mémoire supplémentaire durant l'exécution
 - La mémoire est libérée à la fin du programme
 - *Avantage* : Rapidité, on n'a pas besoin « d'aller chercher » de la mémoire à l'exécution
 - *Inconvénient* : c'est pas très flexible, on doit connaître exactement la taille des données à mettre en mémoire.

L'allocation mémoire 2/3

- L'allocation dynamique sur la pile
 - Seulement la mémoire nécessaire à une procédure (ou fonction) est allouée lors de son exécution
 - Les variables définies dans la procédure sont
 - Allouées lors de l'entrée dans la procédure
 - Libérées (automatiquement) à la sortie

L'allocation de mémoire 3/3

- L'allocation dynamique sur le tas
 - La mémoire est allouée et désallouée au besoin au fur et à mesure du programme dans un pool de mémoire
 - C'est plus flexible
 - On n'a pas besoin de connaître a priori le nombre d'objets à créer, leur durée de vie, leur type
 - Mais c'est plus complexe et plus dangereux
 - Cela nécessite plus de ressources (et de temps) pour allouer de la mémoire de manière dynamique
 - Le programmeur doit libérer la mémoire par un objet qui n'est plus utilisé
 - Et il peut « oublier » de libérer cette mémoire (fuite) ou pire libérer la mémoire plusieurs fois
 - Il existe néanmoins des processus automatisés de libération de la mémoire appelés « ramasse miettes »

Java et l'allocation mémoire

- Java utilise principalement une allocation dynamique sur le tas pour les objets et dispose d'un ramasse-miette (Garbage Collector)
 - Le programmeur n'a pas à se soucier de la libération de la mémoire
- A chaque fois que le programmeur veut créer un objet il utilise le mot clé « new » pour allouer la mémoire

Les erreurs de programmation

- Il existe plusieurs types d'erreurs
 - Erreur de syntaxe : le programme ne peut pas être compilé / s'exécuter
 - Erreur à l'exécution : le programme s'exécute mais plante lors de l'exécution
 - Erreur sémantique : tout semble bien se passer mais le résultat n'est pas celui attendu
- Pour minimiser le risque d'erreur :
 - Les tests
 - Les fonctionnalités des langages : exceptions

Traitement des erreurs via les exceptions

- L'exécution d'un programme peut générer des « erreurs »
 - Division par 0, mémoire insuffisante, débordement d'indice, etc.
- Java fourni un mécanisme pour gérer les erreurs appelées « Exceptions »
- Une exception est un objet
 - Qui est « lancé » de l'endroit où l'erreur s'est produite
 - Et qui doit être « attrapée » et gérée par un intercepteur d'exception

Traitement des erreurs via les exceptions

- Avantages
 - Permet de faire du code plus clair
 - la gestion des exceptions est séparée du code « normal »
 - Et plus sûr
 - Les méthodes déclarent les types d'exceptions qu'elles sont susceptible de lever
 - Et lors d'un appel à cette méthode, le programmeur est obligé de traiter l'exception (et donc il ne l'oublie pas)