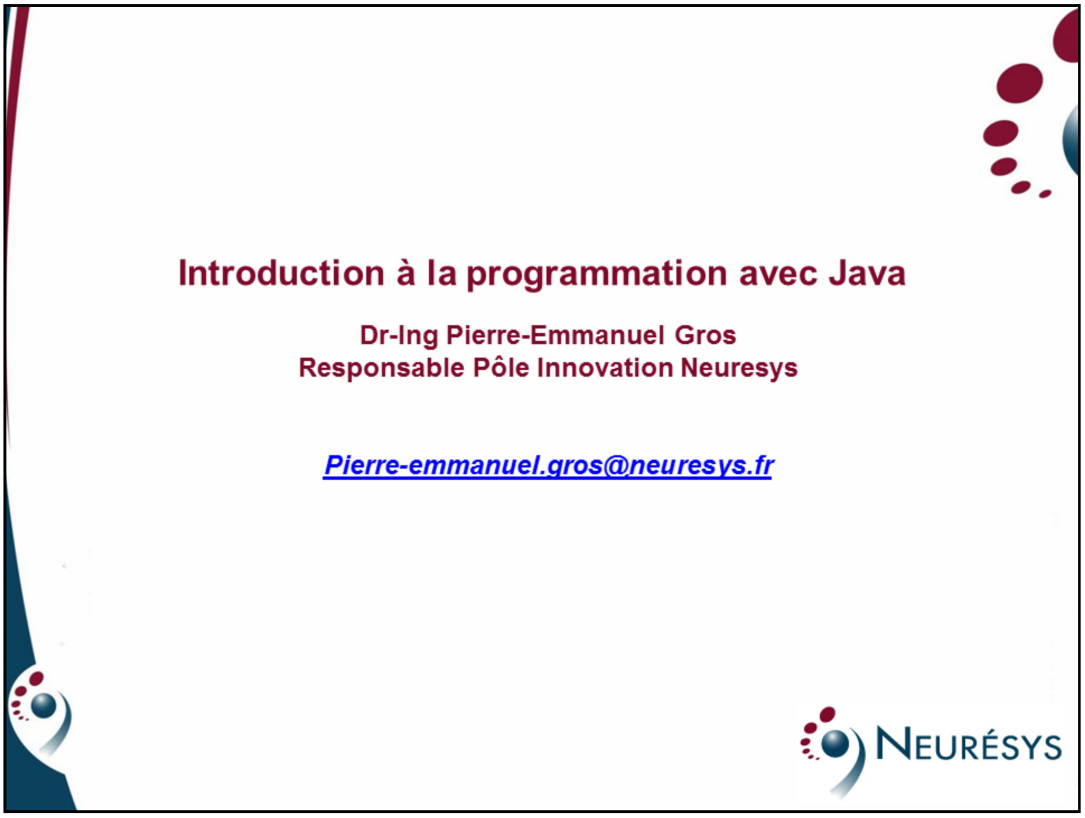

Introduction à la programmation avec Java
Dr-Ing Pierre-Emmanuel Gros
Responsable Pôle Innovation Neuresys
Pierre-emmanuel.gros@neuresys.fr
NEURÉSYS

Introduction à la programmation avec Java

- Les fondements de la programmation
- Genèse d'un premier programme
- Règles de programmation
- Les variables
- Opérateurs et expressions
- Les structures de contrôle
- Les procédures et les fonctions
- Introduction à la programmation objet
- L'accès aux bases de données
- Maintenance, débogage et test des programmes

Introduction à la programmation avec Java

- Les fondements de la programmation
  - Qu'est-ce qu'un programme ? Qu'est-ce qu'un langage ? Les différents paradigmes. Quel langage pour quelle application ?
  - Les compilateurs. Les exécutables.
  - Les responsabilités d'un programmeur.
  - Qu'est-ce qu'un algorithme ?, Les besoins auxquelsrépond un algorithme.
  - Le concept de pseudo-langage.
- Genèse d'un premier programme
- Règles de programmation
- Les variables
- Opérateurs et expressions
- Les structures de contrôle
- Les procédures et les fonctions
- Introduction à la programmation objet
- L'accès aux bases de données
- Maintenance, débogage et test des programmes

Les fondements de la programmation

Qu'est-ce qu'un programme ? Qu'est-ce qu'un langage ? Les différents paradigmes. Quel langage pour quelle application ?

- Contenu
  - Visualisation de « l'écosystème » de la programmation
- Objectif pratiques
  - Connaitre les différents paradigmes informatiques
  - Savoir quand utiliser Java.. Ou pas

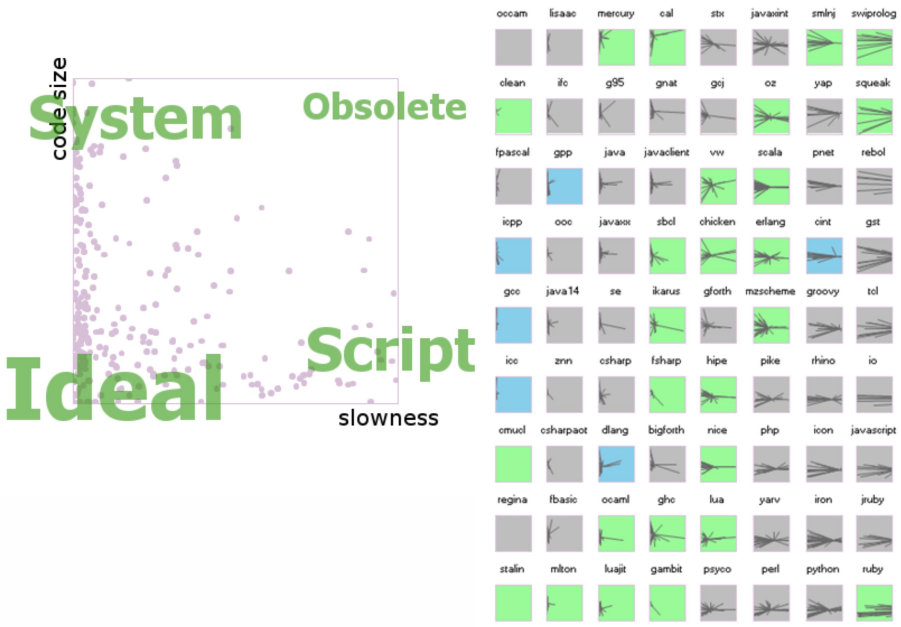
Qu'est-ce qu'un programme ? Qu'est-ce qu'un langage ?

- De façon vulgaire:
  - Un programme est un séquence d'instruction exécuté sur une machine pour obtenir un résultat
  - Un langage est un moyen de définir/créer un programme.
- De façon plus formel
  - un langage de programmation est une notation conventionnelle destinée à formuler des algorithmes et produire des programmes informatiques qui les appliquent. D'une manière similaire à une langue naturelle, un langage de programmation est fait d'un alphabet, un vocabulaire, des règles de grammaire, et des significations. (source Wikipedia)
- De façon plus théorique:
  - Un langage informatique est nommé langage de programmation, s'il permet de représenter toutes les fonctions calculables au sens de Turing et Church (nonobstant la finitude de la mémoire des ordinateurs)

Les paradigmes de programmation

- Programmation impérative, paradigme originel et le plus courant
  - Programmation structurée, visant à structurer les programmes impératifs
  - Programmation procédurale, à comparer à la programmation fonctionnelle
- Programmation orientée objet, consistant en la définition et l'assemblage de briques logicielles appelées objets
  - Programmation orientée prototype, qui simplifie et rend plus flexible la programmation orientée objet
  - Programmation orientée classe
  - Programmation orientée composant (comme en OLE)
- Programmation déclarative, consistant à déclarer les données du problème, puis à demander au programme de le résoudre
  - Programmation descriptive, à l'expressivité réduite, qui permet de décrire des structures de données (par exemple, HTML, XML ou LaTeX)
  - Programmation fonctionnelle, avec laquelle un programme est une fonction au sens mathématique du terme
  - Programmation logique, consistant à exprimer les problèmes et les algorithmes sous forme de prédicats (comme en Prolog)
  - Programmation par contraintes, à comparer à la programmation logique

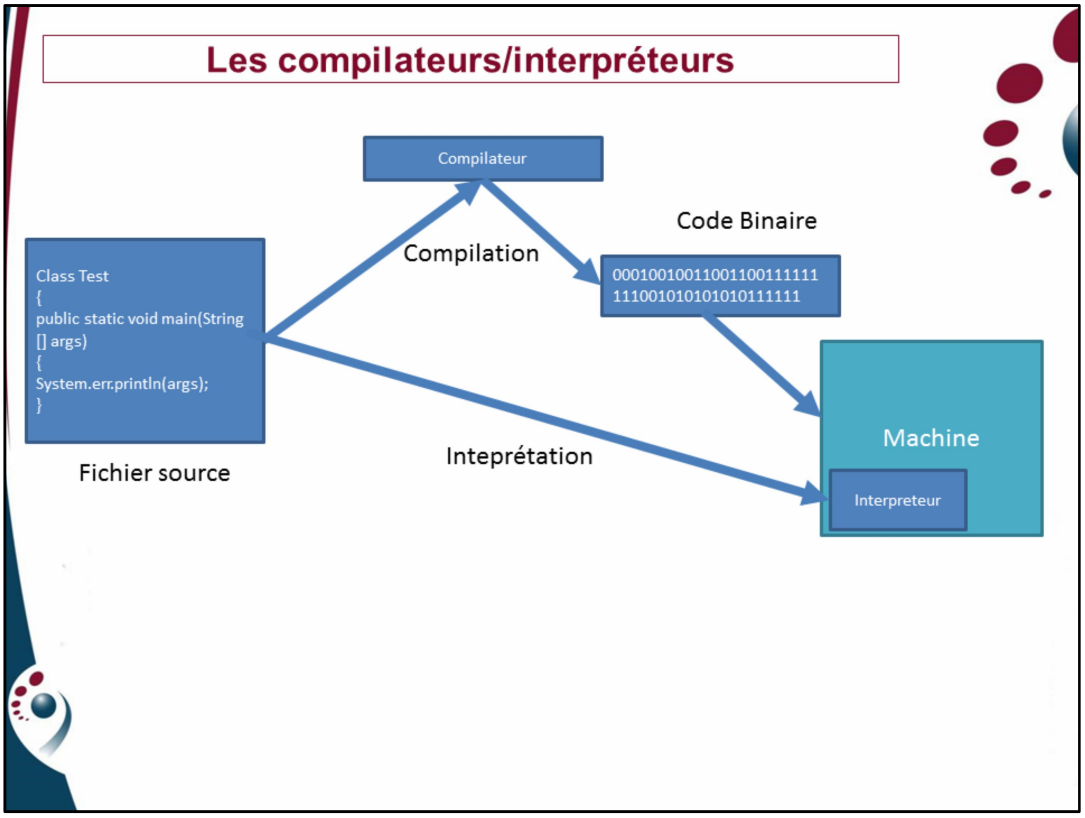
Tentative de comparaison des langages



Les fondements de la programmation

Les compilateurs. Les exécutables.

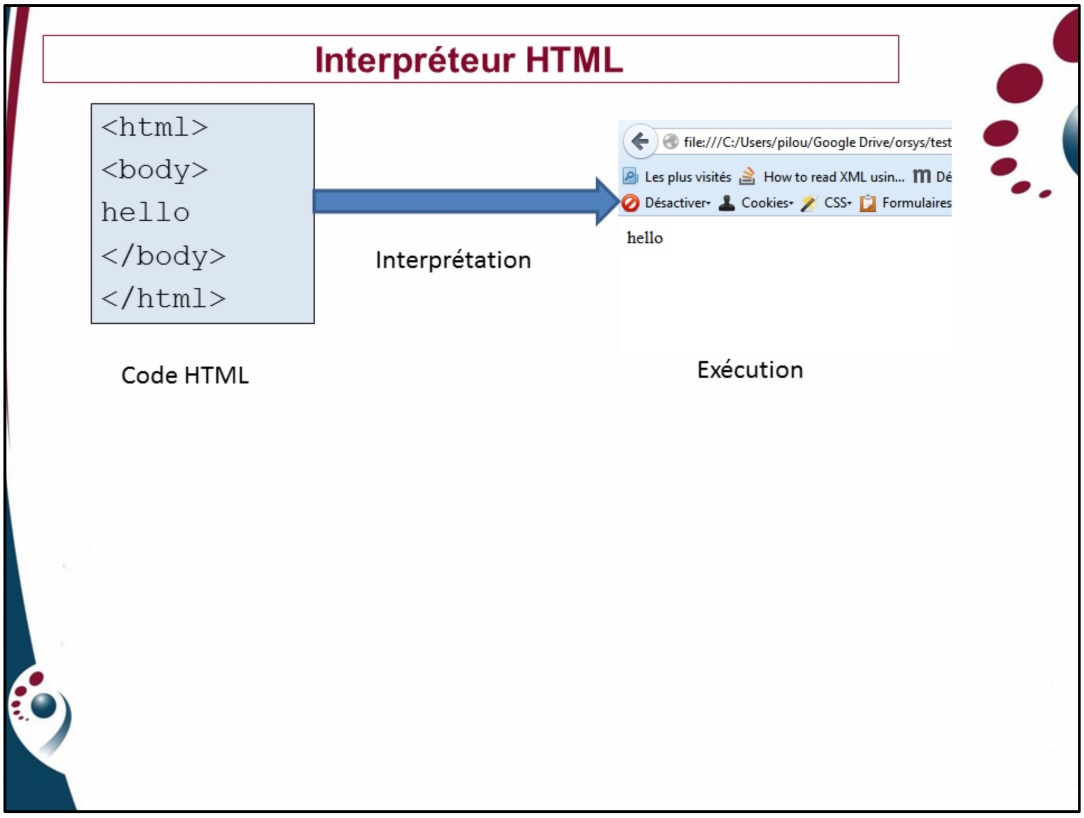
- Contenu
  - Qu'est ce que la chaîne de compilation
- Objectif pratiques
  - Savoir que Java est un langage compilé
  - Savoir ce qu'est un compilateur vs interpreteur


Les compilateurs/interpréteurs
Compilateur
Code Binaire
Compilation
Class Test
{
public static void main(String
[] args)
{
System.err.println(args);
}
0001001001100110011111
1110010101010111111
Machine
Interpreteur
Inteprétation
Fichier source

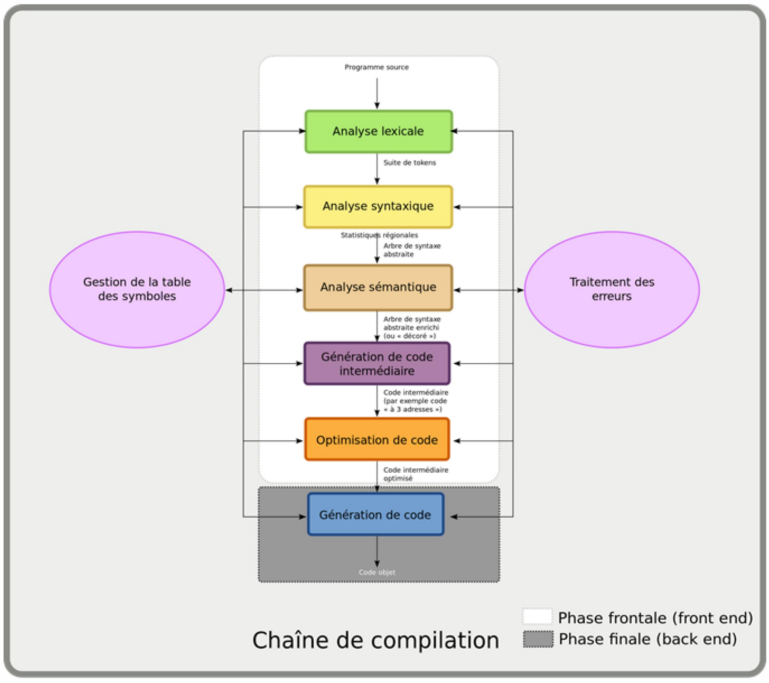
Terminologie

- Une machine est un objet exécutant des instructions. Celle-ci peut être physique (votre ordinateur, smartphone), ou virtuelle. La « machine virtuelle java » est un programme simulant une machine.
- Les sources : fichiers contenant du texte représentant un programme
- Librairie: ensemble de fonctionnalités utilisable dans un programme.
- Compilateur: programme prenant des sources et des librairies pour en faire un programme « executable ».
- Execution: Mise en œuvre d'un programme informatique
- Interpréteur: programme « exécutant » des sources et des librairies pour faire une execution.

Interpréteur HTML



Compilateur ex Java



Interpréteur vs compilateur

	Interpréteur	Compilateur
Performance	-	++
Facilité	++	-
Rapidité de développement	++	-
Sureté de développement	-	++
Utilisé en entreprise	++	++
QQ Exemples	Html, JavaScript, Lua	Java, C#,Lua

Les fondements de la programmation

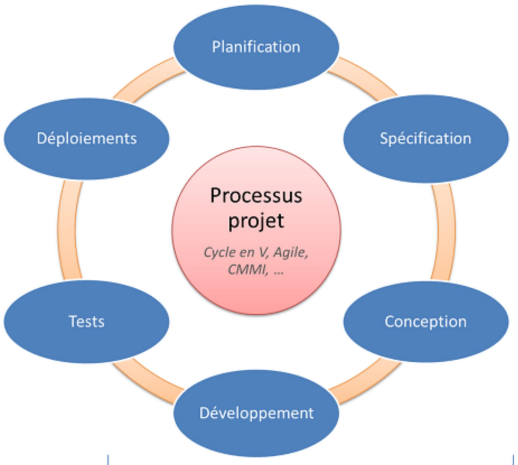
Les responsabilités d'un programmeur.

- Contenu
  - Qu'est ce qu'un développeur
- Objectif pratiques
  - Savoir qu'on attend d'un développeur
  - De façon subjectif, qu'est ce qu'un bon développeur

Un développeur

« est un informaticien qui réalise des fonctionnalités/la maintenance d'un outil informatique. Le profil du développeur est celui d'un technicien ou d'un ingénieur capable d'analyser les besoins des clients consignés au préalable dans un cahier des charges par le chef de projet. Il préconise et met en œuvre une solution technique pour concevoir des outils sur mesure ou adapter des solutions techniques existantes. »

Cycle de vie d'un projet



Le métier de développeur

Qualité d'un developpeur

- Compétences essentielles:
  - Savoir lire et écrire
  - Compétences techniques
  - Prix par jour (entre 300 €HT/J à 600 €HT/J)
  - Qualité de production
- Compétences appréciés:
  - Expertise technique
  - Capacité de communication
  - Capacité d'anticipation

Les fondements de la programmation

Qu'est ce qu'un algorithme

- Contenu
  - Définition d'un algorithme
  - Exemple d'algorithme simple

Algorithme

- « Un algorithme est une suite finie et non ambigüe d'opérations ou d'instructions permettant de résoudre un problème. »
- Propriétés d'un algorithme (D. Knuth):
  - la finitude : « Un algorithme doit toujours se terminer après un nombre fini d'étapes. »
  - définition précise : « Chaque étape d'un algorithme doit être définie précisément, les actions à transposer doivent être spécifiées rigoureusement et sans ambiguïté pour chaque cas. »
  - entrées : « ... des quantités qui lui sont données avant qu'un algorithme ne commence. Ces entrées sont prises dans un ensemble d'objets spécifié. »
  - sorties : « ... des quantités ayant une relation spécifiées avec les entrées. »
  - rendement : « ... toutes les opérations que l'algorithme doit accomplir doivent être suffisamment basiques pour pouvoir être en principe réalisées dans une durée finie par un homme utilisant un papier et un crayon. »

Exemple d'Algorithme

- Non mathématique: les recettes de cuisines, les plans IKEA, les plans LEGO…
- Informatique :
  - Compter les gens dans une salle par itération: « Soit une valeur i valant 0. Tant que la salle n'est pas vide, demander a qq un de sortir de la salle et faire i=i+1 »
  - Compter les gens dans une salle par récurrence.

Service d'un algorithme

- Les problèmes informatiques se répètent souvent de façon identique:
  - Problèmes de trie, jointure…
  - Problématique de graphe: voyageur de commerce, problème de flux, optimisation
  - Problèmes d'IA: solution aux problèmes NP complet

- Les algorithmes sont en fait des briques méthodologiques fiable mise au point par des experts
- L'utilisation d'algorithme permet de solutionner les problèmes de façon efficace en évitant de réinventer la roue

Les fondements de la programmation

Concept de pseudo langage

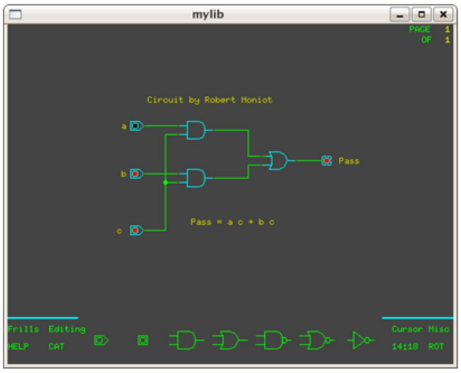
- Contenu
  - Définition d'un pseudo langage
  - Quelques pseudo langage utilisé

Pseudo Langage

- Objet théorique destiné à décrire un algorithme sans se soucier ni du langage de programmation ni de la machine d'execution.
- Généralement relié à des langages de programmation impératif.
- Inusité :
  - Sauf pour des besoins de formations
  - Pour des algorithmes très complexe (recherche).
- Exemple simple:

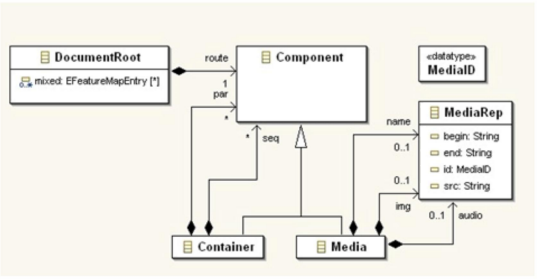
```
soit T le tableau,
soit M = 0
pour tout i du tableau
  Ajouter T[i] dans M
soit res la division de M par la taille du tableau
```

Quelques exemples de pseudo langages



Pseudo langage électronique

Pseudo langage de classe (UML)



Introduction à la programmation avec Java

- Les fondements de la programmation
- Genèse d'un premier programme
  - Compilation et exécution du programme.
  - Ecriture d'un programme simple : syntaxe et instructions.
  - Qu'est-ce qu'une librairie ? Son rôle, son usage
- Règles de programmation
- Les variables
- Opérateurs et expressions
- Les structures de contrôle
- Les procédures et les fonctions
- Introduction à la programmation objet
- L'accès aux bases de données
- Maintenance, débogage et test des programmes

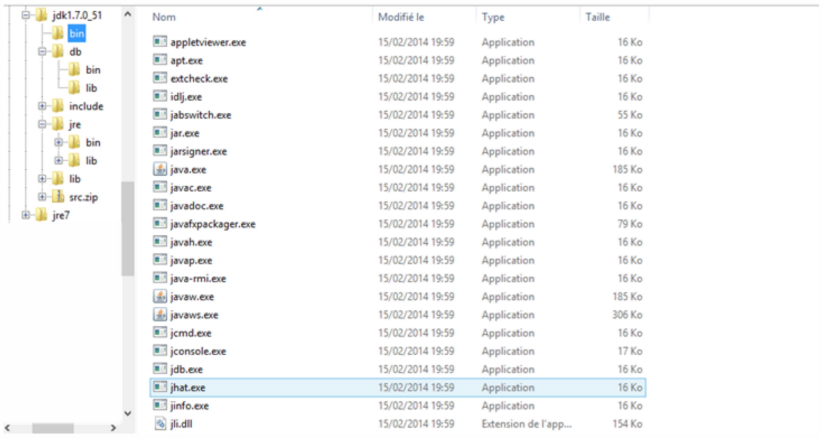
Genèse d'un premier programme

Compilation et exécution du programme.

- Contenu
  - Les outils Java/Javac
  - Créer un programme
- Objectif:
  - Ecrire un programme simple
  - Le compiler
  - L'executer

Java/Javac

- Le langage « Java » est un langage qui se compile avec l'utilitaire « Javac » vers une machine virtuelle « Java ».
- Le compilateur s'appel javac et la machine java



Java/Javac

```
public class test
{
public static void main(String [] args)
{
      System.out.println("Coucou");
}
}
```

```
C:\Users\pilou\>javac.exe test.java
C:\Users\pilou\>dir
15/04/2014  12:56               408 test.class
15/04/2014  12:56               102 test.java
C:\Users\pilou\>java.exe test
Coucou
```

Java/Javac

```
C:\Users\pilou\>javap.exe -c test
Compiled from "test.java"
public class test {
  public test();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method
java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: getstatic     #2                  // Field
java/lang/System.out:Ljava/io/PrintStream;
       3: ldc           #3                  // String Coucou
       5: invokevirtual #4                  // Method
java/io/PrintStream.println:(Ljava/lang/String;)V
       8: return
}
```

Genèse d'un premier programme

Ecriture d'un programme simple.

- Contenu
  - Voir la syntaxe
- Objectif:
  - Comprendre un programme simple

Java

```
/**
On définit une classe appelé test ayant une méthode « static » noté
main
Prennant un tableau de chaîne de caractère (String []) noté args
*/
public class MonPremierProgramme
{
public static void main(String [] args)
{
         /* on demande la classe System
         Ayant un attribut static noté out
         Cet attribut est un objet ayant une méthode
         println
         */
         System.out.println("Coucou");
}
}



C:\java MonPremierProgramme
>Coucou
```

Java

- Le mot « class » portant sur MonPremierProgramme définit un ensemble de fonctionnalité
- Une fonctionnalité appelé « main » est définit (public static void main).

```
/**
On définit une classe appelé test ayant une méthode « static » noté main
Prennant un tableau de chaîne de caractère (String []) noté args
*/
public class MonPremierProgramme
{
public static void main(String [] args)
{
            /* on demande la classe System
            Ayant un attribut static noté out
            Cet attribut est un objet ayant une méthode
            println
            */
            System.out.println("Coucou");
}
}
```

Fonction Main

- La fonction **main** est la fonction principale des programmes en Java: Elle se trouve obligatoirement dans tous les programmes. L'exécution d'un programme entraîne automatiquement l'appel de la fonction **main**.
- Définition de la fonction main « public static void main(String [] args)
- Remarque avancée:
  - Il est possible de faire passer des arguments de la ligne de commande à un programme.

Genèse d'un premier programme

Qu'est ce qu'une librairie

- Contenu
  - Une librairie
  - Les fichiers « jar »
- Objectif:
  - Comprendre ce qu'est une librairie
  - Entrevoir le monde du JRE

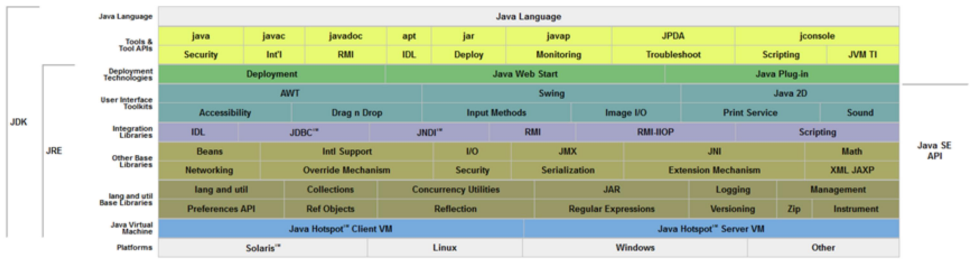
Une librairie

- Un programme Java ou autre est rarement écrit à 100%
- Il s'agit en fait de l'adjonction de composant « pré-écrit » et de 1 à 5% de code « maison ».
- L'idée est qu'un composant pré-écrit est a vécu l'épreuve du temps:
  - Plus robuste (moins d'erreur)
  - Plus facile d'utilisation
  - Plus d'évolutivité

Des librairies

- Il existe 1 librairie particulière qui est le celle du JRE (Java Runtime Environnement) qui est incluse par défaut dans tous les programmes Java.
- Sinon il existe des librairies pour faire de la 3D, de l'image du son, du XML ……
- Lors de l'utilisation d'une librairie X, on parle de l'importation de cette librairie.

Les librairies standard de Java



Utilisation des librairies en Java

- Java.lang : la librairie de « base » de Java avec la classe « System »
- Les autres librairies « standard » s'utilise avec la directive import. Par exemple, pour une liste d'objet on écrie import java.util.ArrayList
- Les autres librairies doivent en plus être précisé à l'éxecution
- La collection des librairies utilisé par un programme est vulgairement appelé « classpath ».

Introduction à la programmation avec Java

- Les fondements de la programmation
- Genèse d'un premier programme
- Règles de programmation
  - Convention de nommage.
  - Convention syntaxique.
  - Utilisation des commentaires. Pourquoi commenter les développements ?
  - Améliorer la lisibilité des programmes: indentation du code, découpage du code...
- Les variables
- Opérateurs et expressions
- Les structures de contrôle
- Les procédures et les fonctions
- Introduction à la programmation objet
- L'accès aux bases de données
- Maintenance, débogage et test des programmes

Règles de programmation

Convention de nommage/syntaxe.

- Contenu:
  - Comprendre l'intérêt d'une convention de nommage
  - Connaitre les conventions de nommage
- Objectif:
  - Connaitre les conventions de Java

Convention de nommage

Chaque objet, classe, programme ou variable est associé à un nom : l'identificateur qui peut se composer de tous les caractères alphanumériques et des caractères _ et $. Le premier caractère doit être une lettre, le caractère de soulignement ou le signe dollar.

Rappel : Java est sensible à la casse.

Un identificateur ne peut pas appartenir à la liste des mots réservés du langage Java :

abstract	const	final	int	public	throw
assert (Java 1.4)	continue	finally	interface	return	throws
boolean	default	float	long	short	transient
break	do	for	native	static	true
byte	double	goto	new	strictfp	try
case	else	if	null	super	void
catch	enum (Java 5)	implements	package	switch	volatile
char	extends	import	private	synchronized	while
class	false	instanceof	protected	this	

Convention de nommage

- Une convention de nommage est un ensemble de règle « commune » pour l'écriture de programme.
- Le respect de ces règles ne sont pas une obligation du langage Java mais une preuve de politesse vis-à-vis des autres développeurs.
- De plus, ces règles ne sont pas là par « hasard ». Elles améliorent la lisibilité et donc diminue les problèmes de programmation.

Convention classique

- Le camelCase c'est la manière la plus utilisée pour nommer en Java.
  - Chaque première lettre d'un mot prend une majuscule
  - tous les mots sont collés les uns aux autres
  - le premier mot ne prend pas de majuscule.

- Un exemple de camelCase est : ceciEstUnExemple.

Convention classique

- Toutes les variables sont en camelCase
- Les noms de classes sont en camelCase avec première lettre en majuscule.
- Les noms de méthodes sont en camelCase.
- Les constantes sont en majuscule
- Les variables i,j,k,l sont souvent associé aux entiers et c,d,e souvent associé au charactere.

Convention classique

```
class Example {
    int[] myArray = { 1, 2, 3, 4, 5, 6 };
    int theInt = 1;
    String someString = "Hello";
    double aDouble = 3.0;

    void foo(int a, int b, int c, int d, int e, int f) {
          switch (a) {
          case 0:
              Other.doFoo();
              break;
          default:
              Other.doBaz();
          }
    }

    void bar(List v) {
          for (int i = 0; i < 10; i++) {
              v.add(new Integer(i));
          }
    }
}
```

Convention classique

- Le symbole { définit une entrée de bloc et est toujours sur la même ligne que la déclaration précédente
- Le symbole } fin de de bloc est toujours précédé d'un retour chariot
- Après chaque { on rajoute une indentation afin de bien voir le bloc. En Java une indentation est soit une tabulation soit 4 espaces.
- En pratique on laisse Eclipse mettre en place la convention de syntaxe

Règles de programmation

Utilisation des commentaires. Pourquoi commenter les développements ?

- Contenu:
  - Qu'est ce qu'un commentaire?
  - Ou positionner les commentaires
- Objectif:
  - Comprendre pourquoi il faut commenter?
  - Comprendre le taux de commentaires.

Les commentaires

- Ils ne sont pas pris en compte par le compilateur donc ils ne sont pas inclus dans le pseudo code. Ils ne se terminent pas par un caractère ";".
- Il existe trois types de commentaire en Java :

Type de commentaires	Exemple
commentaire abrégé	// commentaire sur une seule ligne int N=1; // déclaration du compteur
commentaire multi ligne	/* commentaires ligne 1 commentaires ligne 2 */
commentaire de documentation automatique	/** * commentaire de la methode * @param val la valeur à traiter * @since 1.0 * @return la valeur de retour * @deprecated Utiliser la nouvelle methode XXX */

Commentaires

- Les commentaires sont des textes au format libre aidant à donner du sens a un programme informatique.
- L'idée est de faciliter la relecture, la maintenance et le partage d'information.
- NB: le premier des commentaires est le code source lui-même qui se doit d'être lisible

Quelques soit le commentaire …

```
#include "stdio.h"
#define xyxx char
#define xyyxx putchar
#define xyyyxx while
#define xxyyyx int
#define xxxyyx main
#define xyxyxy if
#define xyyxyy '\n'
xyxx *xyx [] = {
"]I^x[I]k\\I^o[IZ~\\IZ~[I^|[I^l[I^j[I^}[I^n[I]m\\I]h",
"]IZx\\IZx[IZk\\IZk[IZo_IZ~\\IZ~[IZ|_IZl_IZj\\IZj]IZ}]IZn_IZm\\IZm_IZh
",
"]IZx\\IZx[I^k[I\\o]IZ~\\IZ~\\I]|[IZl_I^j]IZ}]I^n[IZm\\IZm_IZh",
"]IZx\\IZx[IZk\\IZk[IZo_IZ~\\IZ~_IZ|[IZl_IZj\\IZj]IZ}]IZn_IZm\\IZm]IZh
",
"]I^x[I]k\\IZo_I^~[I^|[I^l[IZj\\IZj]IZ}]I^n[I]m^IZh",'\0'};/*xyyxyxyxx
xyxxxyy*/
xyxx *xyyx; xxyyyx
xyyyx,xyyyyx,xyyyyyx=0x59,xyyyyyyx=0x29,/*yxxyxyyyxxyyyxyy*/
xxyx=0x68;xxxyyx(){xyyyyx=0;xyyyxx(xyx[xyyyyx]){xyyx=xyx[xyyyyx++];/*x
yyyxxyx*/
xyyyxx(*xyyx){xyyyx= *xyyx++-xyyyyyx;xyyyxx(xyyyx--)xyyxx(*xyyx-
xyyyyyyx);/*x*/
xyxyxy(*xyyx==xxyx)xyyxx(xyyxyy);*xyyx++;}}}/*xyxyxyyyyxxyxxxyyyxyyyxy
xxyyy*/
```

Quoi/Comment commenté

- L'idée est de commenter :
  - Tous les points d'entrée d'une classe (attribut, méthode..)
  - Les parties de code « difficile » (algorithme)
  - Les choix d'implémentations
- Les commentaires doivent être instructif et concie.
- Il ne doivent pas dépasser en nombre de lignes le nombre de lignes de code.

Introduction à la programmation avec Java

- Les fondements de la programmation
- Genèse d'un premier programme
- Règles de programmation
- Les variables
  - Qu'est-ce qu'une variable ?
  - Pourquoi typer une variable ?
  - Les types primitifs : entiers, chaînes de caractères, nombres réels, autres.
  - Déclaration, définition et initialisation d'une variable.
  - Les constantes.
  - Saisie, affichage, affectation, conversion de type.
- Opérateurs et expressions
- Les structures de contrôle
- Les procédures et les fonctions
- Introduction à la programmation objet
- L'accès aux bases de données
- Maintenance, débogage et test des programmes

Les variables

Qu'est ce qu'une variables

- Contenu
  - Description d'une variable
- Objectif:
  - Comprendre ce qu'est une variable
  - Voir la différence entre une variable de méthode et une variable de classe

Les variables

- Un programme Java ou autre utilise de la mémoire de l'ordinateur.
- La mémoire de l'ordinateur est vue par Java comme étant un ensemble de boite portant un nom et pouvant contenir un objet (de type chaussure, balle ….).
- On parle du :
  - nom de la variable pour le nom de la boite
  - Du type de la variable pour le type de la boite

Les variables

- En Java, nous avons donc les classes, les objets et les méthodes (static ou pas).
- Lorsque d'une variable est désigné dans une méthode on parle de « variable ».
- Mais une variable peut être définit pour un objet. On parle alors d'attribut de l'objet.
  - Par exemple l'âge d'un humain est une « variable » de la classe humain qui prendra une valeur lors de l'instanciation de la classe

Les variables

- Une variable possède un nom, un type et une valeur. La déclaration d'une variable doit donc contenir deux choses : un nom et le type de données qu'elle peut contenir. Une variable est utilisable dans le bloc où elle est définie.
- La déclaration d'une variable permet de réserver la mémoire pour en stocker la valeur.
- Le type d'une variable peut être :
  - soit un type élémentaire dit aussi type primitif déclaré sous la forme type_élémentaire variable;
  - soit une classe déclarée sous la forme classe variable ;

```
long nombre;
int compteur;
String chaine;
```

Les variables

- Java est un langage à typage rigoureux qui ne possède pas de transtypage automatique lorsque ce transtypage risque de conduire à une perte d'information.
- Pour les objets, il est nécessaire en plus de la déclaration de la variable de créer un objet avant de pouvoir l'utiliser. Il faut réserver de la mémoire pour la création d'un objet (remarque : un tableau est un objet en Java) avec l'instruction new. La libération de la mémoire se fait automatiquement grâce au garbage collector.
- Il est possible en une seule instruction de faire la déclaration et l'affectation d'une valeur à une variable ou plusieurs variables.

```
MaClasse instance; // déclaration de l'objet


instance = new MaClasse(); // création de l'objet


MaClasse instance = new MaClasse(); // déclaration
//et création de l'objet
```

Les variables

Les types primitifs : entiers, chaînes de caractères, nombres réels, autres.

- Contenu
  - Liste des types de variables
- Objectif:
  - Connaitre quelques « grand » types de variables

Les types de variables

- Une variable se définit comme une boite où l'on va mettre des « objets ».
- Une boite de banane ne peut contenir qu'une banane et une boite d'ananas contient des ananas
- Dire qu'une boite de X ne peut contenir que des X c'est donner la définition du type d'une variable.
- Une boite de X se dit en informatique une variable de type X

Les principaux types primitifs

- Parmis les types de variables :
- Les « booléen » qui peut contenir la valeur « vraie » ou la valeur « fausse »
- Les entiers ou integer qui contiennent des nombres entier (1,2,3 …)
- Les nombres floattant qui contiennent des nombres avec des virgules (1.2, 0.002 …)
- Le type « rien » ou « void » utiliser dans le cadre des fonctions/procedure.

Les principaux types primitifs

- Les types élémentaires ont une taille identique quelque soit la plate-forme d'exécution : c'est un des éléments qui permet à Java d'être indépendant de la plate-forme sur laquelle le code s'exécute.
- Les types élémentaires commencent tous par une minuscule.

Type	Désignation	Longueur	Valeurs	Commentaires
boolean	valeur logique : true ou false	1 bit	true ou false	pas de conversion possible vers un autre type
byte	octet signé	8 bits	-128 à 127	
short	entier court signé	16 bits	-32768 à 32767	
char	caractère Unicode	16 bits	\u0000 à \uFFFF	entouré de cotes simples dans du code Java
int	entier signé	32 bits	-2147483648 à 2147483647	
float	virgule flottante simple précision (IEEE754)	32 bits	1.401e-045 à 3.40282e+038	
double	virgule flottante double précision (IEEE754)	64 bits	2.22507e-308 à 1.79769e+308	
long	entier long	64 bits	-9223372036854775808 à 9223372036854775807	

Les types « objets »

- Le type « objet » est un type définie par l'utilisateur ayant un nom choisie par l'utilisateur.
- Les types « objets » sont des assemblages de boites de type différents.

Les types tableaux

- Il peut être utile de définir qu'une boite peut contenir d'autre boite.
- Par exemple, on veut pouvoir définir une boite spécial qui contiendras d'autre boite de bananes.
- C'est la notion du type tableau qui est un type spécial pouvant contenir des boites.

Les variables

Déclaration, définition et initialisation d'une variable.

- Contenu:
  - Commencer l'usage des variables
- Objectifs:
  - Déclarer une variables
  - Mettre une valeur dans une variables
  - Initialiser une variable
  - Comprendre la valeur null

Déclaration, définition et initialisation d'une variable.

```
Class Test
{
Public static void main(String [] args)
{
// on définit une variable par
// type de variable<espace> nom de variable
}
}
```

Déclaration d'un variable

- un nom de variable ne peut comporter que des lettres, des chiffres (les caractères _ et $ peuvent être utilisés mais ne devrait pas l'être pour des variables)
- Un nom de variable ne peut commencer par un chiffre et comporter d'espace
- Les noms de variables ne peuvent pas être les noms réservés du langage

Déclaration d'une variable.

```
Class Test
{
Public static void main(String [] args)
{
// déclaration d'un variable b de type booléen
boolean b;
// de même pour un entier du nom i
int i;
// ou pour un nombre a virgule
float f;

}
}
```

Définition d'une variable.

```
Class Test
{
Public static void main(String [] args)
{
// déclaration d'un variable b de type booléen
Boolean b;
// définition de sa valeur
b = true;
// de même pour un entier du nom i
Int i;
i=2;
// ou pour un nombre a virgule
Float f;
f=3.3;
}
}
```

Définition d'un variable de type classe/tableau

- Les classes sont de type comme les autres.
- La seul différence entre un type primitif et une classe/tableau est que un type primitif est une simple boite.
- Les tableaux sont des boites de boites.
- Les classes sont des définitions plus complexes de boites (la classe humain contient une boite pour l'age, pour le nom ...).

Définition d'un variable de type classe/tableau

- Lors de la définition d'un tableau, on va donner combien de boite ce tableau peut accueillir.
- Un tableau se définit ainsi:
  - **<TypeSimple> tableau <NomTableau>[<Dimension>]**
  - Ex int tableau1[], ou int tableau2[][]
- L'initialisation de valeur a un tableau se fait ainsi:
  - int tableauEntier[] = {0,1,2,3,4,5,6,7,8,9};
  - int premiersNombres[][] = { {0,2,4,6,8},{1,3,5,7,9} };

Définition d'un variable de type classe/tableau

- Si un tableau n'est pas contruit via une initalisation, il est nécéssaire de préciser sa taille.
- Cette dernière se fait via l'opérateur **new:**
  - int [] a; a=new int [**4**]; **// 4 cases entières sont réservé.**

Définition d'un variable de type classe/tableau

- Les opérations sur les tableaux sont les suivantes:
  - **Soit un tableau X : int [] X=new int [5].**
- Le tableau X a 5 cases noté de 0 à 4.
- Il est possible de connaitre la taille d'un tableau X.length
- Il est possible d'écrire une valeur dans un tableau : X[2]=3;
- Il est possible de lire une valeur dans un tableau int a=X[1];

Définition d'une variable de type classe/tableau

- Une classe est un assemblage de « variable » appelé membres.
- Une structure est un assemblage de variables qui peuvent avoir différents types. Contrairement aux tableaux qui vous obligent à utiliser le même type dans tout le tableau, vous pouvez créer une structure comportant des variables de types long, char, int et double à la fois.
- Une définition de classe se fait en général dans un fichier portant le nom de la classe (la classe Toto seras écrite dans un fichier Toto.java)

Définition d'une variable de type classe/tableau

- Elle se définit par :

public class *Nom de la classe*

{

public « *type de variable* » « *nom de variable* »;

}

Définition d'une variable de type classe/tableau

- L'operateur new est aussi utilisé pour les classe.

```
Public class Humain
        { ...
        ....
        }
Class Test
{
Public static void main(String [] args)
{
        // définition d'un humain
        Humain x;
        // initialisation  d'un humain
        x=new Humain ()
}
}
```

Définition d'une variable de type classe/tableau

- Et on accède aux membres d'une classe via la notation « pointé »

```
Public class Humain
        {
        public int age;
        public String nom;
        }

Public Class Test
{
        Public static void main(String [] args)
{
        // définition d'un humain
        Humain x;
        // initialisation d'un humain
        x=new Humain ();
        x.age=3;
}
}
```

Définition d'une variable de type classe/tableau

- En pratique les tableaux et les classes sont initialisé via l'appel a new.
- Si l'initialisation est oublié, on dit que la valeur est « null » et sont emploie provoque des erreurs (essayer).
- « null » est une valeur particulière sans type signifiant « pas initialisé ».

Les constantes

- Les constantes sont des variables:
  - Appartenant a une définition de classe (utilisation de l'attribut static)
  - Qui ne sont pas modifiables (utilisation de l'attribut final)

```
Class Test
{
      public static final int MACONSTANTE=3;
}
```

Les variables

Saisie, affichage, affectation, conversion de type.

- Contenu:
  - Voir comment saisir des variables
  - Voir comment afficher les variables
  - Regarder la conversion de type
- Objectif:
  - Arriver a lire une ligne taper sur le clavier
  - Convertir une ligne représentant un entier … en entier

Les flux

- Un programme Java est un programme normal avec une gestion de flux :
  - Standard : flux d'entrée (le clavier), flux de sortie et flux d'erreur
  - Spécifique : flux réseau, flux fichier ...

- Pour lire, nous utilisons le flux d'entrée qui est une variable de la classe System nommée in.

Lire un flux

- Nous allons crée un objet Scanner qui permet de lire des variables
- Nous l'initialisons avec le flux d'entrée
- Puis nous demandons le premier entier en utilisant la méthode nextInt();

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

Tester le flux

- Il est possible de tester si ce qui est tapez au clavier est bien un entier avec de demander la conversion en entier

```
Scanner sc = new Scanner(System.in);
If (sc.hasInt())
int i = sc.nextInt();
```

Sécuriser la lecture

- Enfin il est possible que la lecture soit problématique.
- Dans ce cas, nous gérons cela avec des « exceptions » qui sont la pour faire des traitements « exceptionnel » en cas de problèmes

```
Try{
Scanner sc = new Scanner(System.in);
If (sc.hasInt())
int i = sc.nextInt();
} catch (RuntimeException e)
{
}
```

Introduction à la programmation avec Java

- Les fondements de la programmation
- Genèse d'un premier programme
- Règles de programmation
- Les variables
- Opérateurs et expressions
  - Les différents opérateurs (multiplicatif, additif, comparaison, égalité, logique, affectation).
  - Combinaison d'opérateurs.
  - Expression booléenne.
- Les structures de contrôle
- Les procédures et les fonctions
- Introduction à la programmation objet
- L'accès aux bases de données
- Maintenance, débogage et test des programmes

Les différents opérateurs

- Les opérateurs en Java ont une signification différentes en fonction des types de variables sur lequels ils portents.
- Nous avons :
  - L'addition qui portent sur les entiers/floattant et chaine de caractères
  - La soustraction, la multiplication et la division qui portent sur les entiers/floattant
  - L'assignation qui est le fait de mettre une valeur dans une variable
  - Les opérateurs « booléen ».

Opérateur booléen

- Les opérateurs booléens sont des opérateurs dont le résultat est « vrai » ou « faux ».
- Nous trouvons des opérateurs prennant des entiers en paramètres:<, >, <=, >=, == (égalité) ou != différent.
- Il y a aussi des opérateurs prennant en paramètres des valeur booléenes: && (le ET), || (le ou) et le ! (non).
- Enfin il existe un opérateur d'égalité pour les objets, l'opérateur « equals »

Jouons un peu

- 2<5 est une expression correcte renvoyant vrai
- 3<1 est une expression correcte renvoyant faux.
- 2<5 && 3<1 est une expression correcte prennant « vrai » et « faux » pour les combiner avec un ET. Vrai et faux …. C'est faux.
  - En fait seul vrai et vrai donne un résultat vrai
  - Seul faux ou faux donne un résultat faux.
- ! (2<5 && 3<1) est une expression prennant « vrai » et « faux » et calculant l'inverse (avec l'opérateur !)

Introduction à la programmation avec Java

- Les fondements de la programmation
- Genèse d'un premier programme
- Règles de programmation
- Les variables
- Opérateurs et expressions
- Les structures de contrôle
  - Les sélections alternatives (si, si-alors-sinon, sélection cas).
  - Les blocs d'instructions (notion de Début... Fin).
  - Les boucles itératives (tant que-répéter, répéter-jusqu'à, pour-de-à).
  - Imbrication des instructions.
  - Les commentaires.
- Les procédures et les fonctions
- Introduction à la programmation objet
- L'accès aux bases de données
- Maintenance, débogage et test des programmes

Les structures de contrôle

Les sélections alternatives

- Contenu:
  - Selection simple If Then
  - Selection avec cas par défaut: If Then Else
  - Sélection multiple: switch (condition) case (...)
- Objectif:
  - Connaitre et posséder les If Then Else

Les sélections alternatives

- Pour l'instant nous n'avons fait que des programmes executant les instructions les unes après les autres SANS condition.
- Il est possible de mettre en place des conditions afin de faire des traitements différencié. Par exemple un logiciel permettant d'avoir de l'argent dans un distributeur. **Si** le solde de la personne est positif **alors** la machine distribue de l'argent.

Structure de IfThen

- La construction précédente est traduite en Java ainsi:
- If (*Condition booléene*) instruction
- Cela doit être lue en si la condition booléene est vrai alors on execute l'instruction.

Exemple de code

```
int i = 10;

if (i < 0)
  System.out.println("le nombre est négatif");
else
  System.out.println("le nombre est positif");
```

Structure de IfThenElse

- Il est aussi possible de faire une structure Si condition Alors instruction Sinon Instruction.
- La structure s'écrit en Java :
  - If (*condition booléene*) *instruction1* else *instruction2*.
  - A comparer avec If (*condition booléene*) *instruction1*

Exemple de code

```
int i = 0;
if (i < 0)
{
  System.out.println("Ce nombre est négatif !");
}
else
{
  if(i == 0)
    System.out.println("Ce nombre est nul !");

  else
    System.out.println("Ce nombre est positif !");

}
```

Selection multiple

- Il est possible d'enchainer les If Then Else ainsi:
  - If (condition 1) { }; if (condition 2) { } ....
- Cet enchainement peut s'écrire pour plus de lisiblité
  - Switch (condition)

    Case X // cas 1

    Case Y // cas 2

Exemple de code

```
int note = 10; //On imagine que la note maximale est 20

switch (note)
{
  case 0:
    System.out.println("Ouch !");
    break;
  case 10:
    System.out.println("Vous avez juste la moyenne.");
    break;
  case 20:
    System.out.println("Parfait !");
    break;
  default:
    System.out.println("Il faut davantage travailler.");
}
```

Equivalence

- Un If Then Else est un If Then avec la clause Else vide
- Ou autrement dit, un If Then Else peut s'écrire avec 2 If Then
- Les structure switch sont des If Then enchainé

C'est ce qu'on appel le « sucre » syntaxique!

Les structures de contrôle

Les boucles itératives

- Contenu:
  - Boucle for
  - Boule while et do
- Objectif:
  - Connaitre et posséder les boucle while

Boule while

- C'est la boucle tant que qui doit ce lire
  - Tant que la condition est vrai alors faire un bloc d'instruction

```
while (/* Condition */)
{
 //Instructions à répéter
}
```

Exemple de code

```
Int i=0;
While (i<10)
{
System.out.println(i);
i=i+1;
}
```

Boucle faire tant que

- C'est la boucle « sœur » de la boucle while:

```
do{
  //Instructions
}while(a < b);
```

- Contrairement à la boucle while, la boucle do s'execute au moins une fois.

Exemple de code

```
Int i=0;
Do
{
System.out.println(i);
i=i+1;
} while (i<10);
```

Boucle for

- La boucle for est à la fois la plus « simple » et peu devenir la plus complexe.
- Elle se compose de trois partie:
  - La première est la déclaration/initialisation des variables
  - La deuxième est la condition d'arret de la boucle
  - La troisième est l'action a effectuer a chaque tour de boucle.

La boucle for en version simple

```
for(int i = 1; i <= 10; i++)
{
  System.out.println("Voici la ligne "+i);
}
for(int i = 10; i >= 0; i--)
  System.out.println("Il reste "+i+" ligne(s) à
écrire");
```

La boucle for en version complexe

```
// ce type de boucle est à proscrire
for(int i = 0, j = 2; (i < 10 && j < 6); i++,
j+=2){
  System.out.println("i = " + i + ", j = " + j);
}
```

Les structures de contrôle

Imbrication des instructions.

- Contenus
  - Voir la notion d'instruction multiple
  - Comprendre la notion de porté de variable
- Objectifs:
  - Voir la notion de porté

Imbrication des instructions

- Un bloc d'instruction commence par le symbole { et se termine par le symbole }.
- Toutes les instructions de contrôle portent soit sur une instruction soit sur un bloc de variable.
- Par exemple:
  - If (condition) instruction // une seule instruction
  - If (condition) **{ instruction1; instruction2; .... Instruction n; }**

Porté des variables

- Les variables déclarés entre deux accolades ne sont connues qu'entre ces deux accolade.

```
{
int a=0;
if (a<10){System.out.println(a);
}
}
a = 1; // le compilateur refuse. La variable a n'est plus déclaré ici
```

Bonne pratique

- Il est licite de déclarer des blocs d'instructions en dehors de structure de contrôle … mais cela n'améliore pas la lisibilité
- Il est possible de déclarer une instruction dans une structure de contrôle mais cela n'est pas très lisible. On prefereras toujours définir une bloc d'instruction même pour une instruction
  - **int a = 1;{a=a+2;{a=a+3;}}if (a==2) if (a==4) a=a+1;else a++; illisible**
  - **int a = 1;a=a+2;a=a+3;if (a==2) {if (a==4) {a=a+1;}else {a++;}} Ok**

Les structures de contrôle

Les commentaires.

- Contenu:
  - Voir les différents types de commentaires
  - Générer une javadoc
- Objectif:
  - Connaitre l'utilisation des commentaires
  - Connaitre quelques tag javadoc

Commentaires

- Il y a deux syntaxes pour l'utilisation des commentaires
  - // qui permet de faire des commentaires sur une lignes
  - /*
    qui permet de faire des commentaires sur plusieurs lignes
    */
- La syntaxe « classique » en Java consiste a utiliser:
  - La notation // pour des annotations
  - La notation /**
    * pour les autres commentaires
    */

Quoi commenter?

- En pratique on commente les éléments précédé par le mot clef « public »
- La raison est que ce sont les « points » d'entrée des programmes.
- De ce fait la façon de faire de la documentation est « standardisé » en Java via l'outil Javadoc et les tag Javadoc

Tag JavaDoc

```
/**
 * Renvoie l'addition de deux entiers
 * @param a le premier entier
 * @param b le deuxième
 * @return la somme des deux
 */
protected static int addition(int a,int b)
{
      return a+b;
}
```

Introduction à la programmation avec Java

- Les fondements de la programmation
- Genèse d'un premier programme
- Règles de programmation
- Les variables
- Opérateurs et expressions
- Les structures de contrôle
- Les procédures et les fonctions
  - Définitions : procédure, fonction.
  - Pourquoi sont-elles incontournables en programmation (réutilisabilité, lisibilité…) ?
  - Le passage de paramètres.
  - Notion de passage par adresse.
- Introduction à la programmation objet
- L'accès aux bases de données
- Maintenance, débogage et test des programmes

Les procédures et les fonctions

Définitions : procédure, fonction.

- Contenu:
  - Définition d'une procédure
  - Définition d'une fonction
- Objectif:
  - Bien comprendre la notion de fonction membre ou pas

Définition

- une fonction ou une procédure est un « petit » morceau de programme que l'on souhaite utiliser plus d'une fois.
- Une fonction et une procédure peut prendre des « arguments » et renvoie potentiellement quelque chose.
- La syntaxe est celle-ci :

```
Public static <type de retour de la fonction> nom_fonction (parametres ....)
{
Instructions...

return <variable du type du retour de la fonction>;
}
```

Définition

```
public class Humain
	{
	public static int mapremieremethode()
		{

			return 2;
		}
	public static void madeuxièmemethode()
		{
			return;
		}
		public static void main(String []
args)
			{
			}
	}
```

Définition

- Lorsqu'une fonction/methode doit renvoyer « void », le mot clef return est optionnel

```
public static void madeuxièmemethode()
                {
                        //return;
                }
```

Définition

- Il est possible de passer en parametre d'une fonction/methode des arguments.
- En particulier, un objet peut être passer, ici dans la classe Humain, on passe un « Humain » du nom h à la fonction « mapremierermethode »

```
public class Humain
	{
	public static int mapremieremethode(Humain h)
		{
			System.out.print(h);
			return 2;
		}
```

Définition

- Dans ce cas précis :
  - Paramètre de classe d'une fonction appartenant à la classe
- Il est alors possible de supprimer le mot « static » de la fonction. Le paramètre h est alors implicite dans la fonction et appelé « this »

```
public class Humain
        {
        public static int mapremieremethode(Humain h)
                {
                        System.out.print(h);
                        return 2;
                }
        public int mapremieremethode()
                {
                        System.out.print(this);
                        return 2;
                }
```

Les procédures et les fonctions

Pourquoi sont-elles incontournables en programmation.

- Contenu:
  - Comprendre pourquoi c'est plus simple d'utiliser des méthodes
- Objectif:
  - Définir une méthode et voir son intérêt.

Intérêt des methodes

- L'idée derrière les méthodes est de ne pas recopier du code bêtement.
- Supposons un code permettant de faire un calcul complexe. Il peut être utile de le conserver pour le reutiliser.
- La mise en place de méthode implique aussi un code plus petit, avec plus de tests et plus rapide

Intérêt des methodes

- La modularisation du code avec les méthodes est a comparer avec la modularisation du code avec les librairies.
- Il s'agit d'une question de granularité :
  - Les librairies permettent de rendre le code modulaire en mode « gros » grain.
  - Les méthodes rendent le même service en mode grain fin.

Les procédures et les fonctions

Le passage de paramètre.

- Contenu:
  - Comprendre la notion de paramètre
  - Voir comment marche le passage d'un paramètre
- Objectif:
  - Comprendre comment définir les paramètres d'une méthode

Définition

```
public static int max(int a, int b)
            {
                  if (a>b)
                  {
                        return a;
                  }else
                  {
                        return b;
                  }
            }
```

Le passage de paramètres

- Comme vue dans l'exemple, le passage de paramètre est un ensemble de valeur passer a une méthode/fonction/procédure lui permettant de faire son calcul.
- Par exemple, la fonction max prennant deux entiers et renvoyant le plus grand des deux a besoin de deux entiers. On dit que ces deux entiers sont les paramètres de la fonctions.
- Les paramètres peuvent être de tous types et en nombre illimité.
- Enfin une fonction se définit pas son nom et par ses paramètres.

Définition

```
public int max(int a, int b)
            {
                  if (a>b)
                  {
                        return a;
                  }else
                  {
                        return b;
                  }
            }
public int max(int a, int b, int c)
            {
            return max(max(a,b),c);
            }
```

Exemple précédent

- Dans l'exemple précédent, nous definissons une méthode max prennant 3 paramètre de type entier.
- Cette dernière est différente de la méthode max prennant 2 paramètres même si cette dernière a le même nom.
- Enfin nous appelons la méthode max avec deux paramètre dans notre nouvelle méthode.

En pratique comment cela marche?

```
//Regardons ce code
            public class Humain
            {
                        int age;
            }
            protected static void addone(Humain a)
            {
                        a.age=a.age+1;
            }
            protected static void addone(int a)
            {
                        a=a+1;
            }
            public  static void main(String [] args)
            {
                        int entier=0;
                        addone(entier);
                        System.err.println("entier vaut "+entier);
                        Humain moi=new Humain();
                        moi.age=0;
                        addone(moi);
                        System.err.println("entier dans humain vaut "+moi.age);
            }
```

```
entier vaut0
entier dans humain vaut1
```

Passage de paramètre

- En java, les types atomiques sont passé par « valeur ». Ceci veut dire que le paramètre prend la valeur de ce qui est passé par l'appelant.
- De ce fait pour les types atomiques, il faut voir cela comme une copie de la valeur. Dans le corps de la méthode on modifie donc la copie et non pas l'original.
- Pour les objets la situation est différentes. L'objet est passé par référence. C'est-à-dire que pour des raisons de performances Java n'effectue pas une copie de l'objet mais un alias. C'est ce qu'on appel le passage par « référence ».
- De ce fait les objets « paramètres » modifier, modifie l'objet d'origine

Introduction à la programmation avec Java

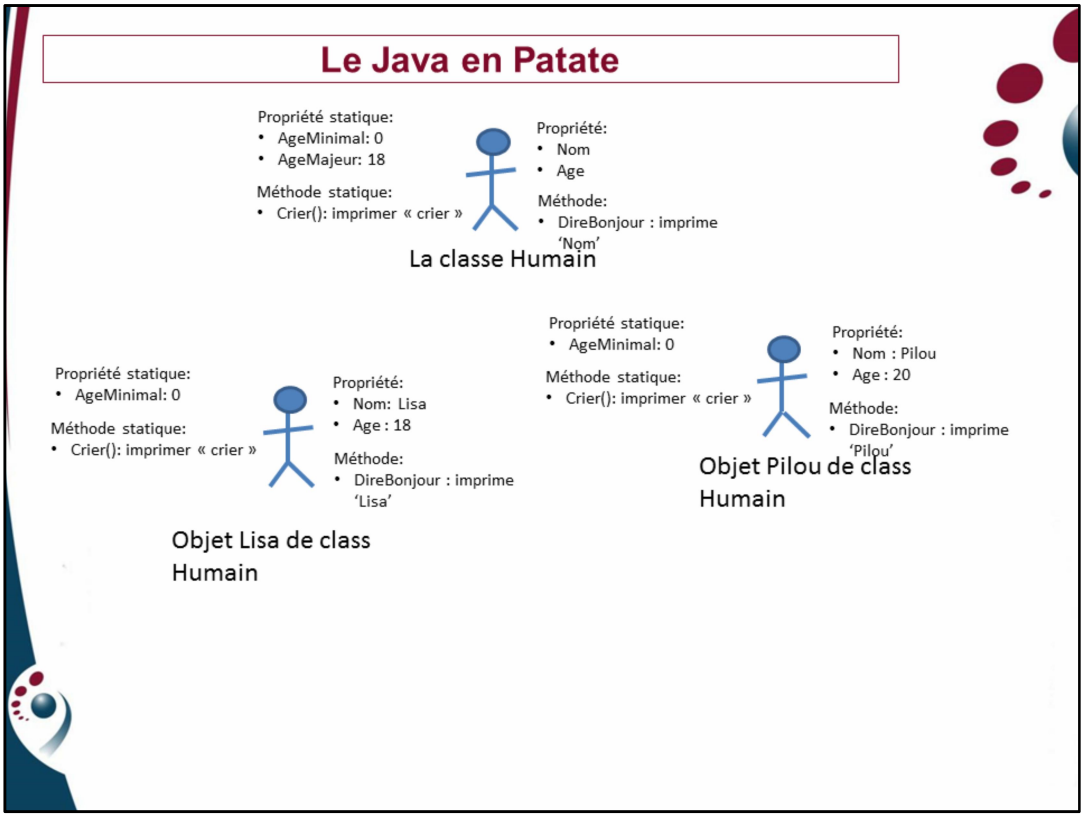
- Les fondements de la programmation
- Genèse d'un premier programme
- Règles de programmation
- Les variables
- Opérateurs et expressions
- Les structures de contrôle
- Les procédures et les fonctions
- Introduction à la programmation objet
  - Les concepts associés à la programmation objet : classe, attribut, méthode, argument.
  - La modélisation objet à partir des exigences fonctionnelles.
  - Introduction aux bonnes pratiques d'organisation de conception et d'organisation d'un programme.
- L'accès aux bases de données
- Maintenance, débogage et test des programmes

Introduction à la programmation objet

Les concepts associés à la programmation objet : classe, attribut, méthode, argument.

- Contenu:
  - Une révision de ce qu'est une classe, attribut, méthode
  - Une révision de la notion de static
  - Présentation de l'héritage
  - Présentation de la notion d'interface
- Objectif:
  - Comprendre l'héritage et la surcharge de méthode

Le Java en Patate



Java

- Pour comprendre Java, il faut voir la notion de « class ».
- Une « class » est une décrit un ensemble de fonctionnalité. Ces fonctionnalité sont appelé « méthode »
- Une instance de class est appelé un objet.
- Un exemple:
  - Nous pouvons décrire un humain. Cette description est appelé class, et nous parlerons de la classe humain.
  - Un etre humain (vous, moi ...) est une « instance » d'humain. En l'occurrence le professeur est une instannce de la classe humain.

Java

- Une classe a des attributs. Par exemple la classe Humain peut avoir comme attribut le nom de l'humain.
  - On peut dire que moi est une instance de la classe Humain dont la valeur de l'attribut « nom » est « Pierre ».
- La plupart des attributs n'ont de valeur que pour un objet (le nom d'un humain, son age ….), par contre certains attributs ont la même valeur pour tous les objets. On dit que ces attributs sont « static ». (Par exemple l'age minimal d'un humain est 0 quelque soit l'humain. L'age minimal est donc un attribut static de la classe humain.
- Une fonctionnalité (on dit méthode) d'une classe porte la plupart du temps sur un objet

Java

- Par exemple, une fonctionnalité de la classe humain peut permettre d'avoir le nom d'un humain. Cette fonctionnalité peut s'appeler « Donne moi ton nom ».
- Parfois certaine fonctionnalité n'appartiennent a aucun objet et son intrinsèque à la classe. Par exemple, la fonctionnalité permettant d'avoir l'age minimal d'un humain est une fonctionnalité de la classe Humain. Ces fonctionnalité sont dit « static »

Java

- Nous avons donc :
  - Les classes qui sont des descriptions contenant des attributs et des méthodes
  - Les objets qui sont des instances de classes.
  - Les attributs et les méthodes prennent le plus souvent sens pour un objet mais peuvent n'avoir de sens que pour les classes. Il sont dit static.
- Soit une classe A ayant un attribut static B, On écrira A.B
- Soit une classe A ayant une méthode static B sans paramètre, On écrira A.B ()
- Soit une classe A ayant un méthode static B prenant 1 parametre (ici 3) , On écrira A.B(3)
- Soit une classe A ayant un attribut static B ayant lui-même une méthode C, On écrira A.B.C()

Les concepts associés à la programmation objet

- Nous avons déjà vue ce qu'est une classe:
  - Une classe est une définition de structure composé d'attribut.
  - Un objet est l'instanciation d'une classe
  - Les méthodes sont les fonctions des classes
  - Les méthodes/attributs sont static (appartenant à la classe) ou pas (elles nécéssitent alors un objet).
- Nous allons voir:
  - L'héritage ou comment définir qu'une classe est « comme une autre » plus quelque chose
  - Les protections
  - Les interface

Les concepts associés à la programmation objet

- Soit la classe Animal ayant une méthode fait du bruit:
  - Nous voulons implémenté la classe Chien qui est un animal mais qui renvoie ouafouaf dans la méthode faitduBruit
  - Nous voulons aussi implémenté la classe oiseau qui non seulement fait « cuicui » mais qui possède une méthode permettant de savoir combien de temps il peut voler.

```
public class Animal {

      public String faitduBruit()
      {
            return "du bruit";
      }
}
```

Les concepts associés à la programmation objet

- Pour la classe Chien, nous allons dire:
  - Qu'un chien est comme un animal. Il s'agit de la notion d'extension ou d'héritage
  - Qu'un chien lorsqu'il fait du bruit, fait ouafouaf. C'est ce qu'on appel la surcharge

- Remarque: un chien est un animal!!

```
class Chien extends Animal
{
          public String faitduBruit()
          {
                    return "ouafouaf";
          }
}
public static void main(String [] args)
          {
                    Animal animal=new Chien();
                    System.out.println(animal.faitduBruit());
// renvoie ouafouaf
          }
```

Les concepts associés à la programmation objet

- Nous pouvons maintenant crée un objet oiseau qui:
  - Fait cuicui
  - Qui possède un constructeur qui est la méthode d'initialisation des objets

```
class Oiseau extends Animal
{
          public int nbvol;
          public Oiseau(int param_nbvol)
          {
                    nbvol=param_nbvol;
          }
          public String faitduBruit()
          {
                    return "cuicui";
          }
}
```

Les concepts associés à la programmation objet

- En Java, il est possible de n'hériter que d'une seul classe
- On peut redéfinir des méthodes, rajouter des méthodes, rajouter des attributs
- Enfin, il est possible de rendre une méthode « abstraite », qui force les classes « héritantes » à l'implémenter.

```
abstract class Oiseau extends Animal
{
            public int nbvol;
            public Oiseau(int param_nbvol)
            {
                        nbvol=param_nbvol;
            }
            public abstract boolean peutVoler();
            public String faitduBruit()
            {
                        return "cuicui";
            }
}
```

Exemple de méthode abstraite

```
class Poulet extends Oiseau
{
// Ce constructeur est forcé car Oiseau
// a un constructeur prenant un entier
        public Poulet(int param_nbvol) {
                super(param_nbvol);
        }
// Cette méthode est forcé car déclarer abstract dans Oiseau
        public boolean peutVoler() {
                return false;
        }
}
```

Interface

- Enfin, une classe n'ayant pas d'attribut et uniquement des méthodes abstraite est dites une interface.

```
interface ObjetVolant
{
        public void vol();
}
class Poulet extends Oiseau implements ObjetVolant
{

        public Poulet(int param_nbvol) {
                super(param_nbvol);
        }
        public boolean peutVoler() {
                return false;
        }
        public void vol() {

        }
}
```

Thread

Initiation au multi-tache en Java

- Comprendre ce qu'est un Thread?
- Commprendre comment faire un thread.

Thread

- les threads sont différents des processus :
  - ils partagent code, données et ressources : « processus légers »
  - mais peuvent disposer de leurs propres données.
  - ils peuvent s'exécuter en "parallèle"
- Avantages :
  - légèreté grâce au partage des données
  - meilleures performances au lancement et en exécution
  - partage les ressources système (pratique pour les I/O)
- Utilité :
  - puissance de la modélisation : un monde multithread
  - puissance d'exécution : paralèllisme
  - simplicité d'utilisation : c'est un objet Java (java.lang)

Thread

- La classe java.lang.Thread permet de créer de nouveaux threads
- Un thread doit implémenter obligatoirement l'interface Runnable
  - **le code exécuté se situe dans sa méthode run()**
- 2 méthodes pour créer un Thread :
  - **1) une classe qui dérive de java.lang.Thread**
    - **java.lang.Thread implémente Runnable**
    - **il faut redéfinir la méthode run()**
  - **2) une classe qui implémente l'interface Runnable**
    - **il faut implémenter la méthode run()**

Extension d'un Thread

```
class Proc1 extends Thread {
Proc1() {...} // Le constructeur
...
public void run() {
... // Ici ce que fait le processus : boucle infinie
}
}
...
Proc1 p1 = new Proc1(); // Création du processus p1
p1.start(); // Demarre le processus et execute p1.run()
```

Implementation d'un Thread

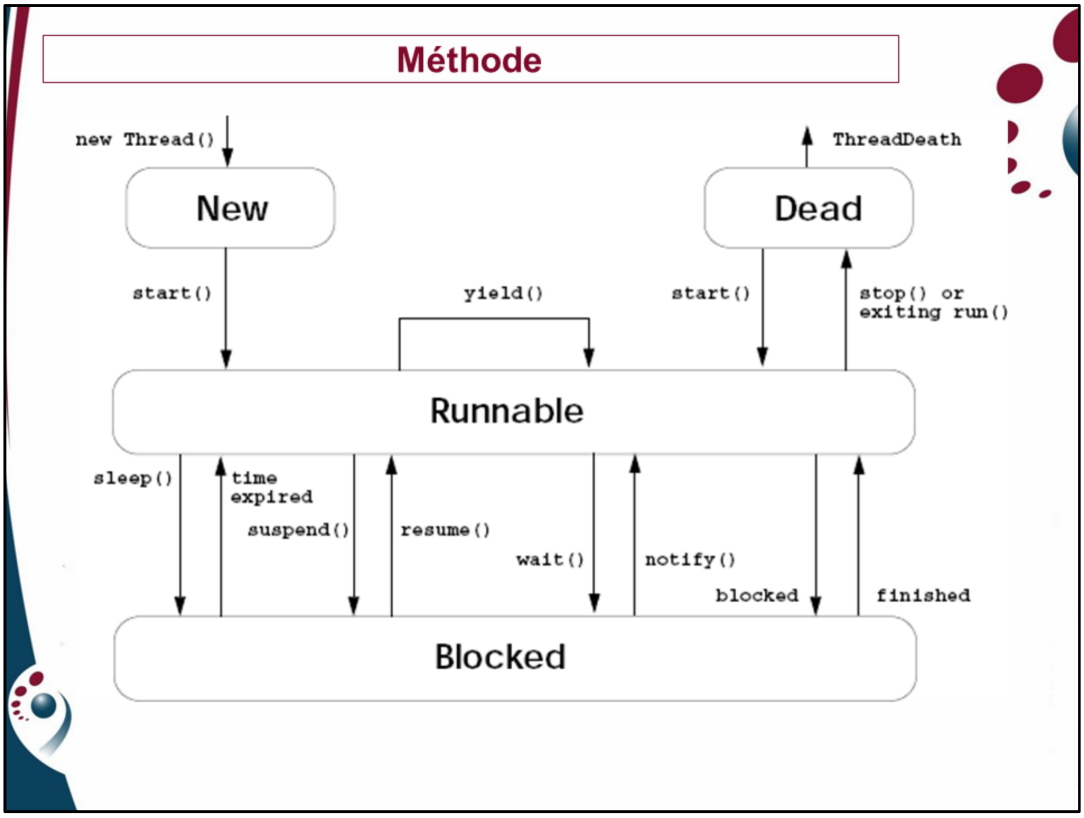
```
class Proc2 implements Runnable {
Proc2() { ...} // Constructeur
...
public void run() {
... // Ici ce que fait le processus
}
}
...
Proc2 p = new Proc2();
Thread p2 = new Thread(p);
...
p2.start(); // Démarre un processus qui execute p.run()
```

Qu'elle est la bonne méthode

- Méthode 1 : sous-classer Thread
  - **lorsqu'on désire paralléliser une classe qui n'hérite pas déjà d'une autre classe (attention : héritage simple)**
  - **cas des** applications autonomes
- Méthode 2 : implémenter Runnable
  - **lorsqu'une super-classe est imposée**
  - **cas des** applets

```
public class MyThreadApplet
extends Applet implements Runnable {}
```

- *Distinguer la méthode run (qui est le code exécuté par l'activité) et la méthode start (méthode de la classe Thread qui rend l'activité exécutable) ;*
- *Dans la première méthode de création, attention à définir la méthode run avec strictement le prototype indiqué (il faut redéfinir Thread.run et non pas la surcharger).*


Méthode
new Thread()
New
ThreadDeath
Dead
start()
yield()
start()
stop() or
exiting run()
Runnable
sleep()
time
expired
suspend()
resume()
wait()
notify()
blocked
finished
Blocked

Creation

- Créé :
  - comme n'importe quel objet Java
  - ... mais n'est pas encore actif
- Actif :
  - après la création, il est activé par start() qui lance run().
  - il est alors ajouté dans la liste des threads actifs pour être exécuté par l'OS en temps partagé
  - peut revenir dans cet état après un resume() ou un notify()

Exemple

```
class ThreadCompteur extends Thread {
int no_fin;
ThreadCompteur (int fin) {no_fin = fin;} // Constructeur
// On redéfinit la méthode run()
public void run () {
for (int i=1; i<=no_fin ; i++) {
System.out.println(this.getName()+":"+i);} }

public static void main (String args[]) {
// On instancie les threads
ThreadCompteur cp1 = new ThreadCompteur (100);
ThreadCompteur cp2 = new ThreadCompteur (50);
cp1.start();
cp2.start();
} }
```

Etats d'un Thread

- Endormi ou bloqué :
  - après sleep() : endormi pendant un intervalle de temps (ms)
  - suspend() endort le Thread mais resume() le réactive
  - une entrée/sortie bloquante (ouverture de fichier, entrée clavier) endort et réveille un Thread
- Mort :
  - si stop() est appelé explicitement
  - quand run() a terminé son exécution

Utilisation d'un sleep

```
class ThreadCompteur extends Thread {
int no_fin; int attente;
ThreadCompteur (int fin,int att) {
          no_fin = fin; attente=att;}
// On redéfinit la méthode run()
public void run () {
for (int i=1; i<=no_fin ; i++) { System.out.println(this.getName()+":"+i);
          try {sleep(attente);}
                    catch(InterruptedException e) {};}
}
public static void main (String args[]) {
// On instancie les threads
ThreadCompteur cp1 = new ThreadCompteur (100,100);
ThreadCompteur cp2 = new ThreadCompteur (50,200);
cp1.start();
cp2.start();
} }
```

Synchronization

- Basée sur la technique de l'exclusion mutuelle :
  - à chaque objet Java est associé un « verrou » géré par le thread quand une méthode (ou un objet) synchronized est accédée.
  - garantit l'accès exclusif à une ressource (la section critique) pendant l'exécution d'une portion de code.
- Une section critique :
  - une méthode : déclaration précédée de synchronized
  - une instruction (ou un bloc) : précédée de synchronized
  - un objet : le déclarer synchronized
- Attention à l'inter-blocage !!
  (problème du dîner des philosophes)

Synchronization

- Pour gérer la concurrence d'accès à une méthode :
  - si un thread exécute cette méthode sur un objet, un autre thread ne peut pas l'exécuter pour le même objet
  - en revanche, il peut exécuter cette méthode pour un autre objet

```
public synchronized void maMethode() {...}
```

- Pour Contrôler l'accès à un objet :

```
public void maMethode() { ...
    synchronized(objet) {
        objet.methode();}}
```

  - l'accès à l'objet passé en paramètre de synchronized(Object) est réservé à un unique thread.

Synchronization

```
class Impression {
synchronized public void imprime(String t) {
for (int i=0; i<t.length(); i++) { System.out.print(t.charAt(i));
 } } }

class TPrint extends Thread {
static Impression mImp = new Impression();
String txt;
public TPrint(String t) {txt = t;}

public void run() {
for (int j=0; j<3; j++) {mImp.imprime(txt);}}

static public void main(String args[]) {
TPrint a = new TPrint("bonjour ");
TPrint b = new TPrint("au revoir ");
a.start();
b.start();
}}
```

Introduction à la programmation objet

La modélisation objet à partir des exigences fonctionnelles

- Contenu:
  - Une approche de UML comme outils à la conception
- Objectif:
  - Connaitre la terminologie UML
  - Voir comment se l'approprier

Problématique de modélisation

- Lorsqu'on débute, il est difficile de voir pour un projet complexe comment prendre la situation.
- Il est possible d'utiliser quelques diagrammes issue des travaux UML afin de faciliter:
  - La discussion avec les utilisateurs (que doit faire le produit)
  - La discussion avec les développeurs (comment le produit est fait).
  - La mise en place de la documentation
- En pratique, sauf cas spécifique, les diagrammes UML sont utilisé comme aide à la conception.

Diagramme simple

- Il y a trois diagrammes utiles et « simples » pour débuter:
  - Le diagramme de cas d'usage qui doit montrer ce que doit faire un utilisateur avec le produit
  - Le diagramme de classe plus technique qui montrent les différentes classes, methodes ainsi que leurs interractions
  - Le diagramme de séquence qui tente de rassembler « lier » le cas d'usage avec le diagramme de classe
- Un exemple classique est celui du distributeur de banque:
  - Il est possible de rentrer un code personnel
  - De choisir un montant d'argent à récupérer (20, 40, 50 et 100 euros)
  - De récupérer sa carte soir après avoir récupérer l'argent soit avant

Diagramme d'usage

- On represente le système (un carré)
- Les usages (des ronds)
- Des types d'usagers (les humains)
- On fait un lien entre les usages et les usagers

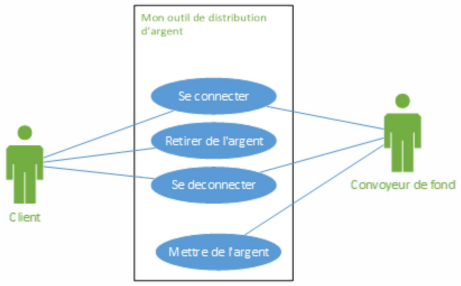


Diagramme de classe

- On essaye de représenter les classes java comme des carré avec leur méthode.
- On représente un trait entre deux classes pour associer les classes.

Un diagramme de classe plus complexe

- Les flèches représente l'héritage entre classe
- Les flèches pointillé les implémentation d'interface

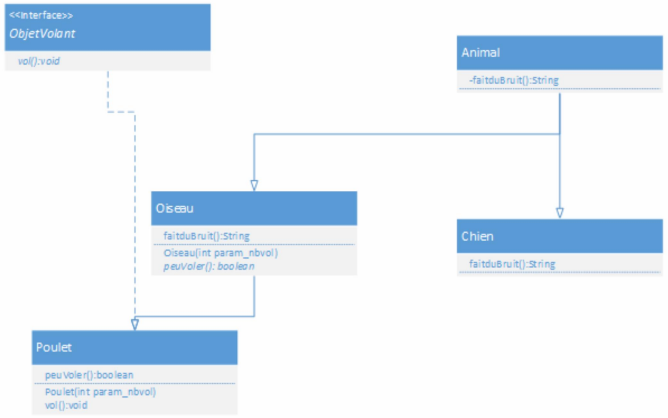
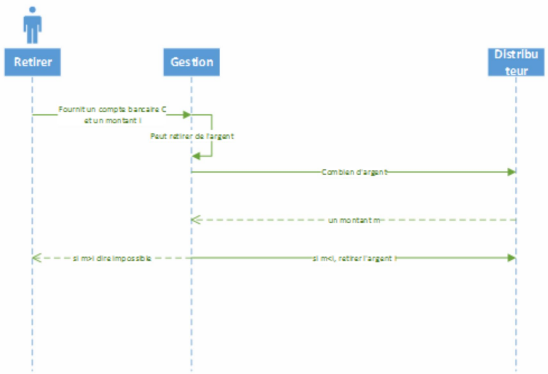


Diagramme de séquence

- Pour chaque usage, on tente de mettre en relation les différentes classses.
- La première barre est soit un usage soit une classe
- La temporalité va de gauche à droite



Introduction à la programmation objet

Introduction aux bonnes pratiques d'organisation de conception et d'organisation d'un programme.

- Contenu:
  - Comment organiser un programme
  - Quelques elements de « syntaxes »
- Objectif:
  - Architecturer un programme

Comment architecturer un programme

- Un programme Java contient rapidemment beaucoup de fichiers que cela soit des sources, des binaires, des librairies…

- L'idée général est de voir comment organiser pour pouvoir se retrouver.
- De se donner quelques règles d'écriture assez commune

Organisation des répertoires

- En général on trouve comme répertoire:
  - Src contenant les sources du programmes
  - Bin ou Jar contenant un fichier Jar (un zip) des binaires compilés
  - Optionnellement un repertoire class contenant les sources compilés mais non agrégé
  - Un repertoire doc contenant les fichiers de documentation
  - Un répertoire thirdparties contenant les librairies annexes utilisés.

Organisation d'une classe

- On définit des classes dans des packages qui sont la dénomination du répertoire où sont les sources.
  - Par exemple les fichiers qui sont dans le répertoire src\com\neuresys\utils sont définie dans un package com.neuresys.utils.
- On ne définit pas de classes à la racine du repertoire src mais toujours un package
- On importe uniquement les classes dont on a besoin.

Nomination des classes

- Une interface définit en fait un concept (par exemple ObjetVolant) et doit donc porter un nom de concept
- Les classes sont des implémentations dites « concrète » qui en général:
  - Rappel le nom de l'interface si il y a une implémentation (par exemple PouletObjetVolant)
  - Porte parfois le suffice Impl pour montrer qu'il s'agit d'une implémentation assez générique d'une interface (ce qui donne ObjetVolantImpl)
  - Porte en préfixe le mot Abstract si une des méthodes de la classes est abstraites

Nomination des méthodes

- Si le but d'une méthode est uniquement de renvoyé une propriété, cette méthode est préfixé par « get »
  - Pour les propriétés booléenes, une tel méthode est préfixé par is
- Si le but d'une méthode est de mettre a jour une propriété, cette méthode est préfixé par set
- Enfin les noms de méthodes essaye d'être explicite sur ce que font ces méthodes

Introduction à la programmation avec Java

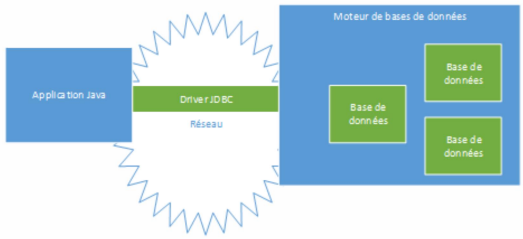
- Les fondements de la programmation
- Genèse d'un premier programme
- Règles de programmation
- Les variables
- Opérateurs et expressions
- Les structures de contrôle
- Les procédures et les fonctions
- Introduction à la programmation objet
- L'accès aux bases de données
- Organisation et stockage des données.
  - Application cliente et serveur de données.
  - Les traitements de base (connexion, requêtes, récupération des données).
  - Affichage et manipulation des données dans l'application cliente.
- Maintenance, débogage et test des programmes

Organisation et stockage des données

Application cliente et serveur de données.

- Contenu:
  - Voir l'organisation client-base de données
  - Comprendre que la base de donnée n'est pas en Java
- Objectifs:
  - Connaitre quelques moteurs de bases de données?
  - Comprendre les termes techniques d'architectures Java/Base de données

La relation Java-Base de données



- Un moteur de base de donnée contient plusieurs bases de données
- Une application Java utilise une librairie particulière nommée driver JDBC pour se connecter à un moteur de base de données
- Le moteur de base de données est adressable par TCP/IP (localisation réseau + numéro de port)

La relation Java Base de données

- Les bases de données utilise la plupart du temps le langage SQL.
- Il faut donc :
  - Se connecter à la base de données en utilisant la librairie JDBC
  - Envoyer des ordres en langage SQL
  - Récupérer des données

Eléments de base

- Il faut un driver (JDBC)
- Il faut une base de donnée
- Une connexion est une relation à la base de données qu'il faut ouvrir… et fermer.
- Un « statement » modélise une requête normé au format SQL
- Un resultSet est un résultat de requête.

L'accès aux bases de données

Les traitements de base (connexion, requêtes, récupération des données).

- Contenu:
  - Comment se connecter a une base de données?
  - Comment créer des données?
  - Comment récupérer des données?
- Objectifs:
  - Créer une base et des données
  - Aborder les API JDBC

Les tables, et les données

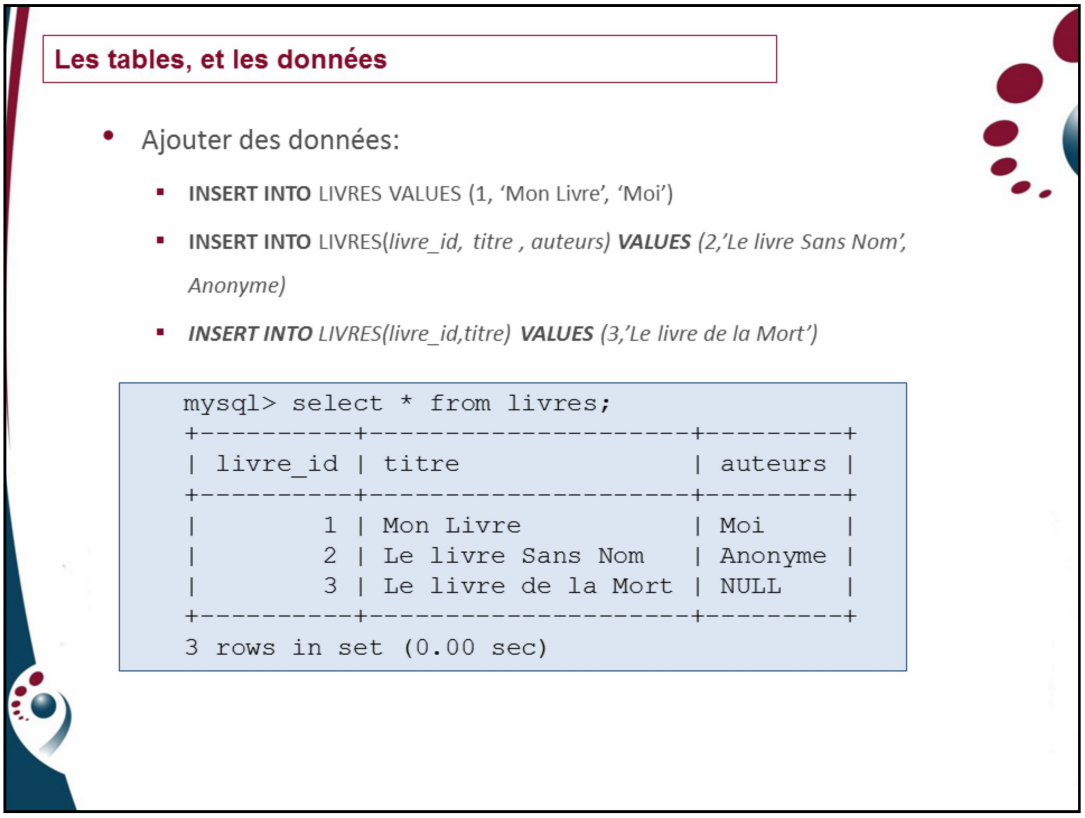
- Connexion à la base:
  - **mysql -h hote -u utilisateur -p** *<Nom de base >*

  Ex (mysql –h localhost –u root –p *<Nom de base>)*
- Création d'une base de données:
  - ***CREATE DATABASE*** *<Nom de base >*
- Utilisation de la base :
  - ***USE*** *<Nom de base >*

Les tables, et les données

- Création d'une base de donnée
  - Create database mabase
  - Use mabase
- Création d'une première table:
  - ***CREATE TABLE*** *livres*

```
(
livre_id INT, # l'identifiant d'un livre
titre VARCHAR(50), -- le titre d'un livre
auteurs VARCHAR(50) /* les auteurs du livre */
);
```



Sssssssss
Sqdqsdqd
Qsdqsdqs
Wfcxwcwxcxw
Qsdqsdqsdqs
Azedazedzaeda

Les tables, et les données

- Sélectionner toutes les données
  - Select * from <nom de table>
- Sélectionner des lignes particulières:
  - Select * from <nom de table> where condition

```
Select * from livres where titre='Mon Livre'
```

- Sélectionner des colonnes particulières
  - Select <nom de colonne> from <nom de table> where condition

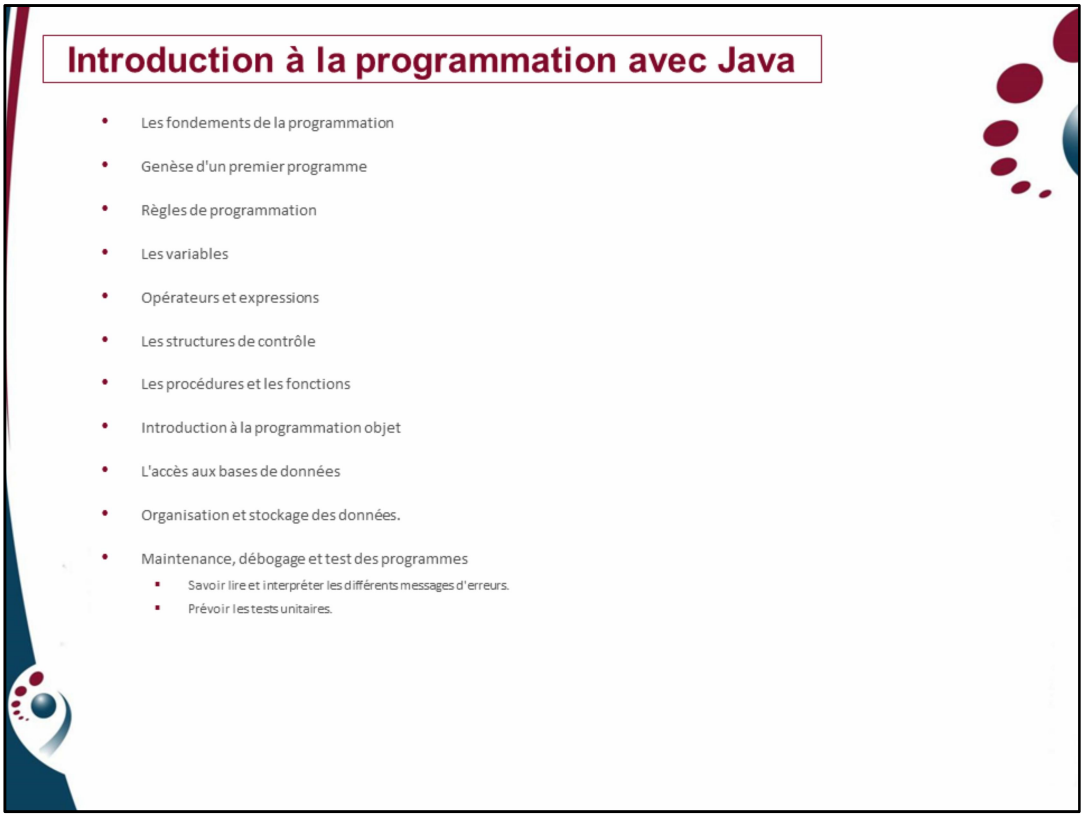
Faire du JDBC simple

```
public static Connection initConnection() throws SQLException, ClassNotFoundException
{
Class.forName("com.mysql.jdbc.Driver");
  return  DriverManager.getConnection("jdbc:mysql://localhost:3306/test","root","");
}
```

```
public static Long findByPrimaryKeyWithoutQueryPlan(Connection c,Long pk) throws
SQLException  {
ResultSet resultSet=null;
Statement statement=null;
try{Long result=0L;
 statement=c.createStatement();
resultSet=statement.executeQuery("select livre_id from LIVRES ");
while (resultSet.next())
{
result=resultSet.getInt(1);
}
return result;
}finally
{
if (resultSet!=null) resultSet.close();
if (statement!=null) statement.close();
}
}
```

Faire du JDBC simple

```
public static void insertData(Connection c,int livre_id, String title,String auteurs) throws
SQLException {
ResultSet resultSet=null;
Statement statement=null;
try{Long result=0L;
 statement=c.createStatement();
statement.executeUpdate("insert into LIVRES VALUES("+livre_id+",'"+title+"','"+auteurs+"')");
return;
}finally
{
if (resultSet!=null) resultSet.close();
if (statement!=null) statement.close();
}
}
```


Introduction à la programmation avec Java
Les fondements de la programmation
Genèse d'un premier programme
Règles de programmation
Les variables
Opérateurs et expressions
Les structures de contrôle
Les procédures et les fonctions
Introduction à la programmation objet
L'accès aux bases de données
Organisation et stockage des données.
Maintenance, débogage et test des programmes
Savoir lire et interpréter les différents messages d'erreurs.
Prévoir les tests unitaires.

Maintenance, débogage et test des programmes

Savoir lire et interpréter les différents messages d'erreurs.

- Contenu:
  - Comprendre les exceptions
  - Lire une stacktrace
- Objectif:
  - Savoir où se trouve une erreurs

Les stack traces

- Il s'agit d'une liste d'appel des méthodes ayant provoqué une erreurs dans un code.
- Le cas simple:

```
Exception in thread "main"
java.lang.NullPointerException
        at
com.example.myproject.Book.getTitle(Book.java:16)
        at
com.example.myproject.Author.getBookTitles(Author.j
ava:25)
        at
com.example.myproject.Bootstrap.main(Bootstrap.java
:14)
```

Les stack traces

- En remontant à la ligne 16 de Book.java, il est probable que title a la valeur null a ce moment

```
public String getTitle() {
    System.out.println(title.toString()); <--
line 16
    return title;
}
```

Les stack traces

- Le cas complexe est celui d'une exception levé a cause d'une autre Exception.
- Le code suivant lance une exception IllegalStateException si une exception NullPointerException est capturé

```
try {
....
} catch (NullPointerException e) {
  throw new IllegalStateException("A book has a
null property", e)
}
```

Les stacks traces

- Dans ce cas la stack traces est représenté ici.
- La partie mise en gras est la cause réel de l'erreur

```
Exception in thread "main" java.lang.IllegalStateException: A
book has a null property
        at
com.example.myproject.Author.getBookIds(Author.java:38)
        at
com.example.myproject.Bootstrap.main(Bootstrap.java:14)
Caused by: java.lang.NullPointerException
        at com.example.myproject.Book.getId(Book.java:22)
        at
com.example.myproject.Author.getBookIds(Author.java:35)
        ... 1 more
```

Maintenance, débogage et test des programmes

Les test unitaires

- Contenu:
  - Savoir ce qu'est un test unitaire
- Objectif:
  - Savoir comment faire un test unitaire

Tests unitaires

- Un test unitaire ou test technique permet de vérifier qu'une méthode fait a peu près ce qu'on attend.
- Autrement dit, soit la méthode max qui prend deux entiers et qui renvoie le plus grand des deux:
  - L'idée d'un test unitaire est de vérifier que le code correspondant à la méthode max fait ce que dit la définition.
  - On va écrire du code pour vérifier ce principe.
  - Cela ne veut pas dire que la méthode fonctionne!!

Test unitaire

- Une façon de faire est de mettre en place des classes de test
- Dans cette classe nous écrivons un « cas de test » (ici 2 et 3)

```
Public class Test
{
Public static int max(int a, int b) ….
Public static void main(String [] args)
{
Int a=2; int b=3;
If (Test.max(a,b)==b) {System.err.println(« test
reussie »);} else{System.err.println(« test
raté »);}
}
}
```

Test unitaire

- Un cas de test ne signifie pas que la méthode fonctionne!!
- Plus une méthode est testé plus elle est « couverte ».
- Dans cette implémentation, le cas b==0 est une erreur!

```
Public class Test
{
Public static int max(int a, int b)
{
If (a/b>1) return a else return b;
}
}
```