OFPPT

Introduction à JAVA

- Introduction
- Syntaxe de base
- La POO
- Les threads

JAVA:Historique

- Java a été développé à partir de décembre 1990 par une équipe de Sun Microsystems dirigée par James Gosling
- Au départ, il s'agissait de développer un langage de programmation pour permettre le dialogue entre de futurs ustensiles domestiques
- Or, les langages existants tels que C++ ne sont pas à la hauteur : recompilation dès qu'une nouvelle puce arrive, complexité de programmation pour l'écriture de logiciels fiables...

JAVA:Historique

- 1990 : Ecriture d'un nouveau langage plus adapté à la réalisation de logiciels embarqués, appelé OAK
  - Petit, fiable et indépendant de l'architecture
  - Destiné à la télévision interactive
  - Non rentable sous sa forme initiale
- 1993 : le WEB « décolle », Sun redirige ce langage vers Internet : les qualités de portabilité et de compacité du langage OAK en ont fait un candidat parfait à une utilisation sur le réseau. Cette réadaptation prit près de 2 ans.
- 1995 : Sun rebaptisa OAK en Java (*nom de la machine à café autour de laquelle se réunissait James Gosling et ses collaborateurs*)

JAVA:Historique

Différentes versions:

- De nombreuses versions de Java depuis 1995
  - Java 1.0 en 1995
  - Java 1.1 en 1996
  - Java 1.2 en 1999 (Java 2, version 1.2)
  - Java 1.3 en 2001 (Java 2, version 1.3)
  - Java 1.4 en 2002 (Java 2, version 1.4)
  - Java 5 en 2004
  - Java 6 en 2006 *celle que nous utiliserons dans ce cours*
  - Java 7 en 2011
- Évolution très rapide et succès du langage
- Une certaine maturité atteinte avec Java 2
- Mais des problèmes de compatibilité existaient
  - entre les versions 1.1 et 1.2/1.3/1.4
  - avec certains navigateurs

JAVA:Historique

- En mai 2007, Sun publie l'ensemble des outils Java dans un « package » OpenJDK sous licence libre.
- La société Oracle a acquis en 2009 l'entreprise Sun Microsystems. On peut désormais voir apparaître le logo Oracle dans les documentations de l'api Java.
- Le 12 avril 2010, James Gosling, le créateur du langage de programmation Java démissionne d'Oracle pour des motifs qu'il ne souhaite pas divulguer.
- Août 2012: faille de sécurité importante dans Java 7
- 2014 : Apparition de Java 8

JAVA:Historique

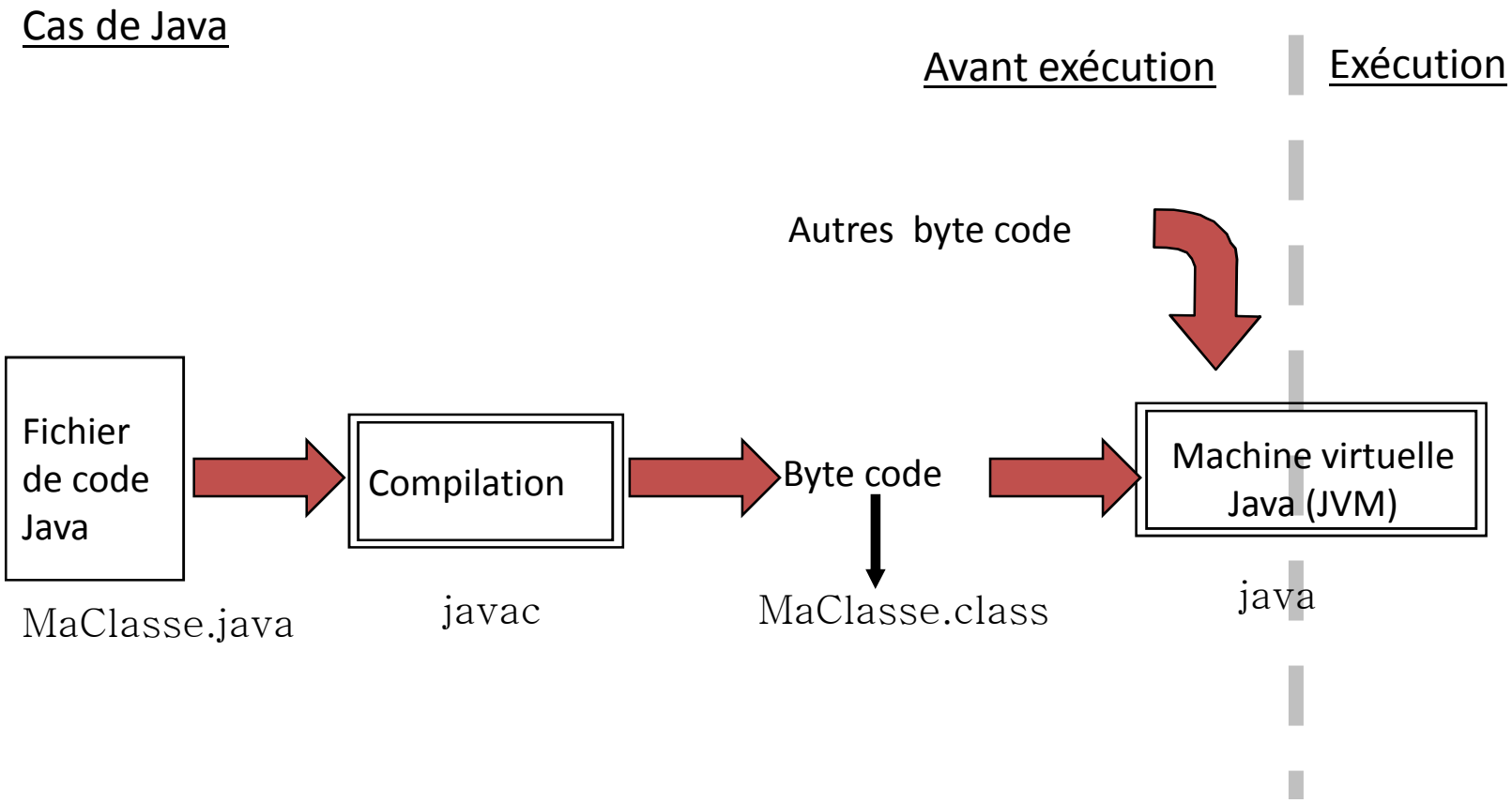
- En mai 2007, Sun publie l'ensemble des outils Java dans un « package » OpenJDK sous licence libre.
- La société Oracle a acquis en 2009 l'entreprise Sun Microsystems. On peut désormais voir apparaître le logo Oracle dans les documentations de l'api Java.
- Le 12 avril 2010, James Gosling, le créateur du langage de programmation Java démissionne d'Oracle pour des motifs qu'il ne souhaite pas divulguer.
- Août 2012: faille de sécurité importante dans Java 7
- 2014 : Apparition de Java 8

JAVA:Caractéristiques

- Simple :Apprentissage facile
- Familier :Syntaxe proche de celle de C/C++
- Orienté objet
- Sûr : Seul le bytecode est transmis, et «vérifié» par l'interpréteur
- Fiable : Gestion automatique de la mémoire *(ramasse-miette ou "garbage collector") ,* Gestion des exceptions , typage fort

- Java est indépendant de l'architecture
- Le bytecode généré par le compilateur est indépendant de toute architecture. Toute application peut donc tourner sur une plate-forme implémentant une machine virtuelle Java *« Ecrire une fois, exécuter partout »*
- Java est multi-tâches

JAVA:Langage interprété



JAVA:JDK

Environnement de développement fourni par Sun
JDK signifie Java Development Kit (Kit de développement Java).
Il contient :

- les classes de base de l'API java (plusieurs centaines),
- la documentation au format HTML
- le compilateur : javac
- la JVM (machine virtuelle) : java
- le visualiseur d'applets : appletviewer
- le générateur de documentation : javadoc
- etc.

JAVA:Synatxe

Commentaire

- /* commentaire sur une ou plusieurs lignes */

  Identiques à ceux existant dans le langage C

- // commentaire de fin de ligne

  Identiques à ceux existant en C++

- /** commentaire d'explication */

  Les commentaires d'explication se placent généralement juste avant une déclaration (d'attribut ou de méthode)

  Ils sont récupérés par l'utilitaire javadoc et inclus dans la documentation ainsi générée.

Main

- Pour pouvoir faire un programme exécutable il faut toujours une classe qui contienne une méthode particulière, la méthode « main »

```
public static void main(String arg[ ])
{
.../...
}
```

JAVA:Synatxe

Identificateurs

- On a besoin de nommer les classes, les variables, les constantes, etc. ; on parle d'identificateur.
- Les identificateurs commencent par une lettre, _ ou $
  - ***Attention : Java distingue les majuscules des minuscules***
- Conventions sur les identificateurs :
  - Si plusieurs mots sont accolés, mettre une majuscule à chacun des mots sauf le premier.
    - exemple : uneVariableEntiere
  - La première lettre est majuscule pour les classes et les interfaces
    - exemples : MaClasse, UneJolieFenetre

JAVA:Synatxe

Les types de bases

- En Java, tout est objet sauf les types de base.
- Il y a huit types de base :
  - un type booléen pour représenter les variables ne pouvant prendre que 2 valeurs (vrai et faux, 0 ou 1, etc.) : **boolean** avec les valeurs associées **true** et **false**
  - un type pour représenter les caractères : **char**
  - quatre types pour représenter les entiers de divers taille : **byte**, **short**, **int** et **long**
  - deux types pour représenter les réelles : **float** et **double**

JAVA:Synatxe

Les Operateurs //même syntaxe en C

- Les structures de contrôle classiques existent en Java **// même syntaxe en C** :
  - **if, else**
  - **switch, case, default, break**
  - **for**
  - **while**
  - **do, while**

JAVA:Synatxe

Les tableaux

- Déclaration
  - int tab [ ];
    String chaines[ ];
- Création d'un tableau
  - tab = new int [20]; // tableau de 20 int
  - chaines = new String [100]; // tableau de 100 chaine
- Création et initialisation simultanées

```
String noms [ ] = {"Boule","Bill"};
Point pts[ ] = { new Point (0, 0), new Point (10, -1)};
```

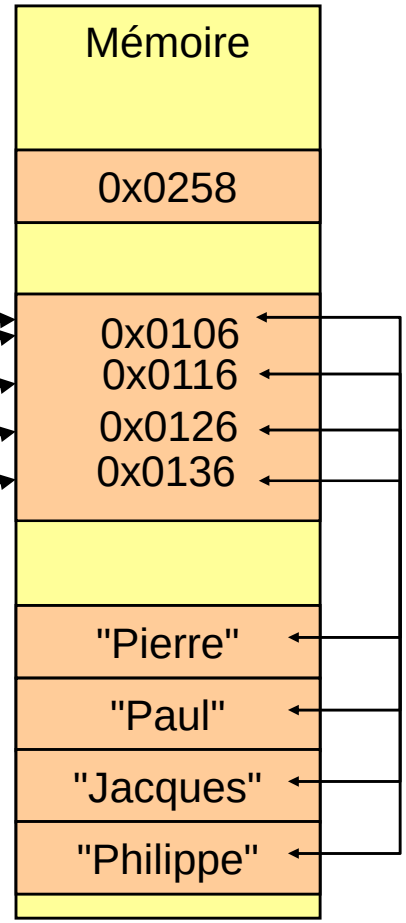
- Le nombre d'éléments : int taille = tab.length;

JAVA:Synatxe

Les tableaux

Tab2.java

```
public class Tab2
{
 public static void main (String args[])
 {
  String tab[ ] ;
  tab = new String[4];
  tab[0]=new String("Pierre");
  tab[1]=new String("Paul");
  tab[2]=new String("Jacques");
  tab[3]=new String("Philippe");
 }
}
```



JAVA:Synatxe

La classe String

- Attention ce n'est pas un type de base. Il s'agit d'une classe défini dans l'API Java (Dans le package java.lang)

  String s="aaa"; // s contient la chaîne "aaa" mais

  String s=**new String**("aaa"); // identique à la ligne précédente

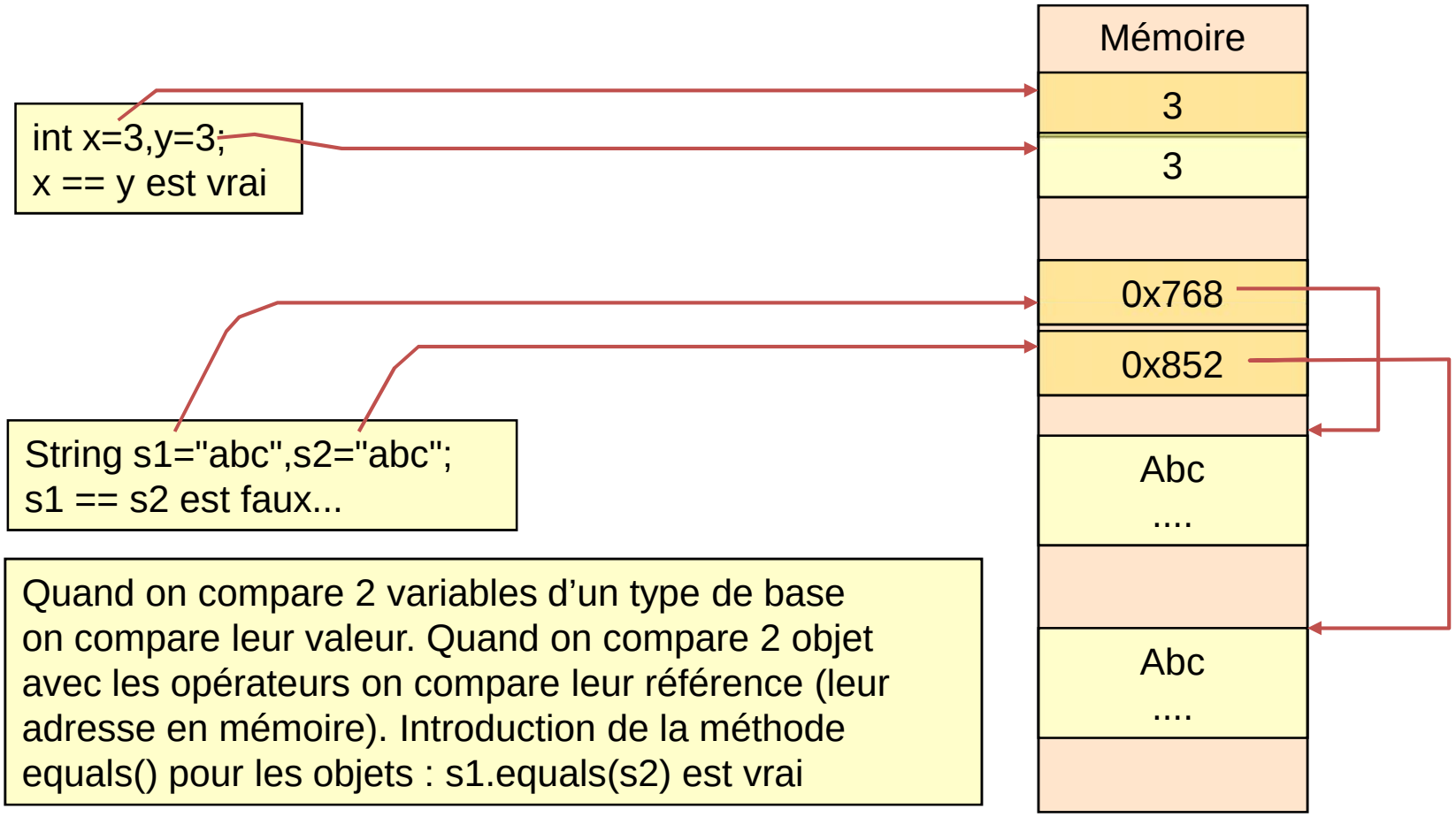
- La concaténation
  - l'opérateur **+** entre 2 String les concatène :

```
String str1 = "Bonjour ! ";
String str2 = null;
str2 = "Comment vas-tu ?";
String str3 = str1 + str2;   / * Concaténation de chaînes : str3 contient " Bonjour ! Comment vas-tu ?
```

- pour tester si 2 chaînes sont égales il faut utiliser la méthode boolean equals(String str) et non **==**
- méthode char charAt(int pos) : renvoie le caractère situé à la position pos

JAVA:Synatxe

La classe String



JAVA:POO

Declaration d'une classe

Exemple : Une classe définissant un point

```
Class Point
{
      double x; // abscisse du point
      double y; // ordonnée du point
      // translate de point de (dx,dy)
      void translate (double dx, double dy) {
            x = x+dx;
            y = y+dy;
      }
      // calcule la distance du point à l'origine
      double distance() {
            return Math.sqrt(x*x+y*y);
      }
}
```

Nom de la Classe

Attributs

Méthodes

JAVA:POO

Les constructeurs

- L'appel de new pour créer un nouvel objet déclenche, dans l'ordre :
  - L'allocation mémoire nécessaire au stockage de ce nouvel objet et l'initialisation par défaut de ces attributs,
  - L'initialisation explicite des attributs, s'il y a lieu,
  - L'exécution d'un constructeur.
- Un constructeur est une méthode d'initialisation

JAVA:POO

Les constructeurs

- L'appel de new pour créer un nouvel objet déclenche, dans l'ordre :
  - L'allocation mémoire nécessaire au stockage de ce nouvel objet et l'initialisation par défaut de ces attributs,
  - L'initialisation explicite des attributs, s'il y a lieu,
  - L'exécution d'un constructeur.
- Un constructeur est une méthode d'initialisation

```
public class Application
{
  public static void main(String args[])
  {
    Personne jean = new Personne()
    jean.setNom("Jean") ;
  } }
```

Le constructeur est ici celui par défaut (pas de constructeur défini dans la classe Personne)

JAVA:POO

Les constructeurs

Personne.java

```
public class Personne
{
  public String nom;
  public String prenom;
  public int age;
  public Personne(String unNom,
                  String unPrenom,
                  int unAge)
  {
    nom=unNom;
    prenom=unPrenom;
    age = unAge;
  }
}
```

Définition d'un Constructeur. Le constructeur par défaut (Personne())n'existe plus. Le code précédent occasionnera une erreur

```
public class Application
{
  public static void main(String args[])
  {
    Personne jean = new Personne()
    jean.setNom("Jean") ;
  } }
```

Va donner une erreur à la compilation

JAVA:POO

Les constructeurs

Personne.java

```
public class Personne
{
  public String nom;
  public String prenom;
  public int age;
  public void Personne()
   {
     nom=null; prenom=null;
     age = 0;
   }
  public String Personne(String unNom,
            String unPrenom, int unAge)
  {
    nom=unNom;
    prenom=unPrenom; age = unAge;
  } }
```

Redéfinition d'un Constructeur sans paramètres

On définit plusieurs constructeurs qui se différencient uniquement par leurs paramètres (on parle de leur signature)

JAVA:POO

Réference

- Lorsqu'une variable est d'un type objet ou tableau, ce n'est pas l'objet ou le tableau lui-même qui est stocké dans la variable mais une référence vers cet objet ou ce tableau (on retrouve la notion d'adresse mémoire ou du pointeur en C).
- Lorsqu'une variable est d'un type de base, la variable contient la valeur.
- Java n'implémente qu'un seul mode de passage des paramètres à une méthode : le passage par valeur.
- Conséquences :
  - l'argument passé à une méthode ne peut être modifié,
  - si l'argument est une instance, c'est sa référence qui est passée par valeur. Ainsi, le contenu de l'objet peut être modifié, mais pas la référence elle-même.

JAVA:POO

Destruction d'objets

- Java n'a pas repris à son compte la notion de destructeur telle qu'elle existe en C++ par exemple.
- C'est le ramasse-miettes (ou Garbage Collector - GC en anglais) qui s'occupe de collecter les objets qui ne sont plus référencés.
- Le ramasse-miettes fonctionne en permanence dans un thread de faible priorité (en « *tâche de fond* »). Il est basé sur le principe du compteur de références.
- Inversement, le ramasse-miettes peut être lancé par une application avec l'appel `System.gc();`

JAVA:POO

Destruction d’objets

- Il est possible au programmeur d'indiquer ce qu'il faut faire juste avant de détruire un objet.
- C'est le but de la méthode `finalize()` de l'objet.
- Cette méthode est utile, par exemple, pour :
  - fermer une base de données,
  - fermer un fichier,
  - couper une connexion réseau,
  - etc.

JAVA:POO

Destruction d'objets

- Il est possible au programmeur d'indiquer ce qu'il faut faire juste avant de détruire un objet.
- C'est le but de la méthode `finalize()` de l'objet.
- Cette méthode est utile, par exemple, pour :
  - fermer une base de données,
  - fermer un fichier,
  - couper une connexion réseau,
  - etc.

JAVA:POO

Modificateurs

- Contrôle d'accès
  - `private`
  - `protected`
  - `public`
  - `package`
- `static` (variables de classe)
- `final` (constantes)
- `transient`
- `volatile`

Les accesseurs (getter ou setter) :

permettent de modifier ou récupérer les attributs d'une classe

Exemple :

```
public String getName() { return name; }
public void setName(String name) { this.name = name; }
```

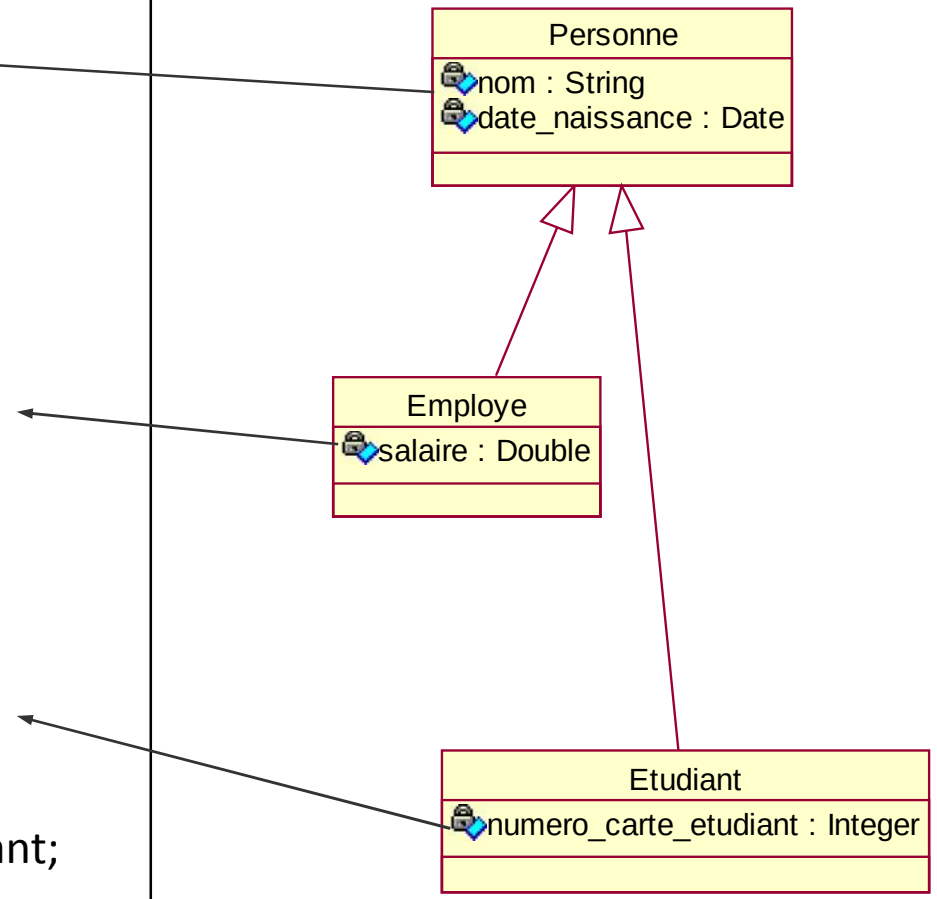
Le mot réservé `this` :

utilisé dans une méthode, désigne la référence de l'instance à laquelle le message a été envoyée (donc celle sur laquelle la méthode est « exécutée »).

JAVA:POO

Héritage

```
class Personne
  {
    private String nom;
    private Date date_naissance;
    // ...
  }

class Employe extends Personne
  {
    private float salaire;
    // ...
  }

class Etudiant extends Personne
  {
    private int numero_carte_etudiant;
    // ...
  }
```



JAVA:POO

Héritage : les constructeur

```
public class Employe extends Personne
{
 public Employe () {}
 public Employe (String nom,
                 String prenom,
                  int anNaissance)
 {
   super(nom, prenom, anNaissance);
 }
}
```

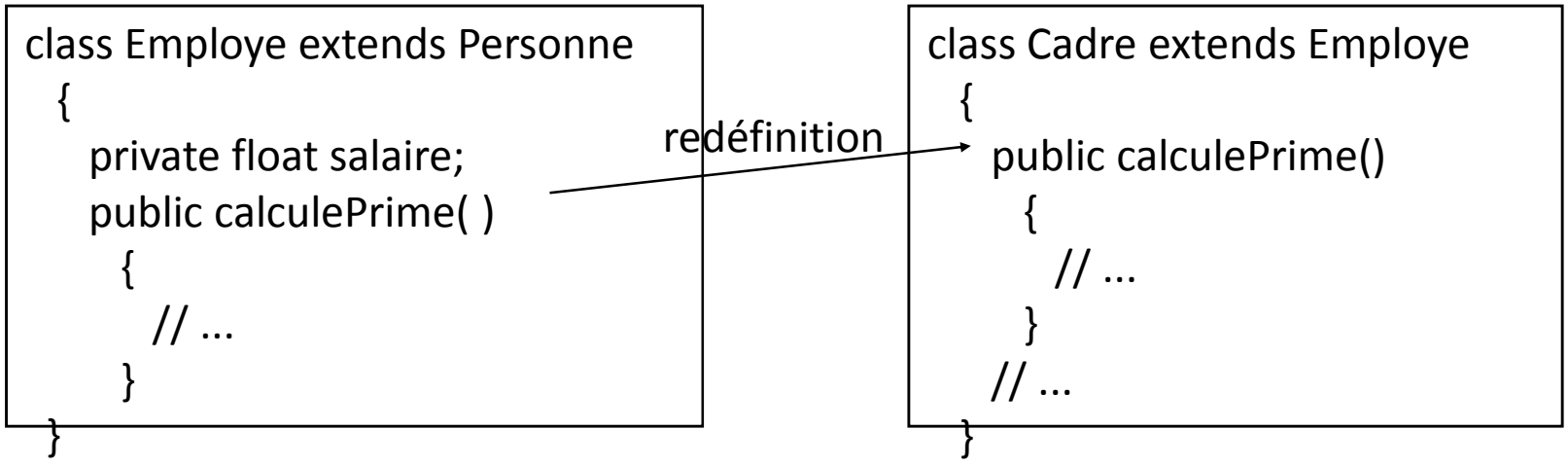
Appel explicite à ce constructeur avec le mot clé super

```
public class Personne
{
   public String nom, prenom;
   public int anNaissance;
   public Personne()
   {
     nom=""; prenom="";
   }
   public Personne(String nom,
                   String prenom,
                   int anNaissance)
   {
     this.nom=nom;
     this.prenom=prenom;
     this.anNaissance=anNaissance;
   }
}
```

JAVA:POO

Redéfinition de methodes :

- Une sous-classe peut redéfinir des méthodes existant dans une de ses superclasses (directe ou indirectes), à des fins de spécialisation.
  - Le terme anglophone est "***overriding***". On parle aussi de **masquage**.
  - La méthode redéfinie **doit avoir la même signature**.



JAVA:POO

Polymorphisme

- Capacité pour une entité de prendre plusieurs formes.
- En Java, toute variable désignant un objet est potentiellement polymorphe, à cause de l'héritage.
- Polymorphisme dit « d'héritage »

- le mécanisme de "lookup" dynamique :
  - déclenchement de la méthode la plus spécifique d'un objet, c'est-à-dire celle correspondant au type réel de l'objet, déterminé à l'exécution uniquement (et non le type de la référence, seul type connu à la compilation, qui peut être plus générique).
  - Cette dynamicité permet d'écrire du code plus générique.

JAVA:POO

Surcharge

- Dans une même classe, plusieurs méthodes peuvent posséder le même nom, pourvu qu'elles diffèrent en nombre et/ou type de paramètres.
  - On parle de **surdéfinition** ou **surcharge**, on encore en anglais d'**overloading** en anglais.
  - Le choix de la méthode à utiliser est fonction des paramètres passés à l'appel.
    - Ce choix est réalisé de façon statique (c'est-à-dire à la compilation).
  - Très souvent les constructeurs sont surchargés (plusieurs constructeurs prenant des paramètres différents et initialisant de manières différentes les objets)

JAVA:POO

- L'opérateur instanceof confère aux instances une capacité d'introspection : il permet de savoir si une instance est instance d'une classe donnée.
- Le transtypage

```
class Personne
  {
    private String nom;
    private Date date_naissance;
    // ...
  }

class Employe extends Personne
  {
    public float salaire;
    // ...
  }

Personne jean = new Employe ();
float i = jean.salaire;  // Erreur de compilation
float j = ( (Employe) jean ).salaire;    // OK
```

JAVA:POO

- Classe abstraite

```
abstract public class FormeGeometrique1 {
double posX, posY;
void deplacer(double x,double y) { posX=x; posY=y; }
void afficherPosition() {
System.out.println("position : (" +posX+","+posY+")"); }

abstract double surface() ;
abstract double perimetre() ;
}
```

- Cette classe est abstraite car ses méthodes surface et perimetre sont abstraites, c.a.d. non définies.
- Une telle classe n'est pas instanciable

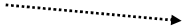
```
public class Rectangle7 extends FormeGeometrique1 {
double largeur, hauteur;
public Rectangle7() { posX=0; posY=0; largeur=0; hauteur=0; }
public Rectangle7(double x, double y, double la, double lo)
 { posX=x; posY=y; largeur=la; hauteur=lo; }
double surface() { return largeur * hauteur; }
 double perimetre() { return 2*(largeur + hauteur); }
}
```

La classe Rectangle7 hérite de la classe abstraite.
Elle définie toutes ses méthodes, en particulier celles abstraites héritées. Donc elle n'est pas abstraite.

JAVA:POO

- Interface :exemple

```
interface Electrique
{
  void allumer();
  void eteindre();
}
```



```
class Radio  implements Electrique
{
  // ...
  void allumer() {System.out.println(‘’bruit ’’);}
  void eteindre() {System.out.println(‘’silence’’);}
```

```
class Ampoule implements Electrique
{
  // ...
  void allumer() {System.out.println(‘’j’éclaire ’’);}
  void eteindre() {System.out.println(‘’plus de lumière’’);}
}
```

JAVA:POO

- Interface :exemple

```
 interface Electrique;
…
interface Lumineux;
…
class Ampoule implements
Electrique, Lumineux
…
Electrique e;
Object o = new Ampoule();
  if (o instanceof Electrique)
  {e=(Electrique)o;e.allumer();}
```

JAVA:POO

Les threads

- 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
- Utilité :
  - puissance de la modélisation : un monde multithread
  - puissance d'exécution : paralèllisme
  - simplicité d'utilisation : c'est un objet Java (java.lang)
- 2 méthodes pour créer un Thread

JAVA:POO

Les threads : Methode 1 : Sous-classer 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()
```

JAVA:POO

Les threads : Méthode 2 une classe qui implémente Runnable

```
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()
```

JAVA:POO

Les threads : Interruption et reprise d'un Thread

- On peut interrompre un thread par l'intermédiaire de l'opération « ***suspend*** ».

- Pour relancer l'exécution d'un thread, on fait appel à la méthode « ***resume*** ».

- On peut également marquer une pause dans l'exécution d'un thread en employant l'opération « ***sleep*** ».

- Enfin, un thread peut attendre la fin d'un autre thread en appliquant l'opération « ***join*** » sur le thread en question.
- Pour arrêter un thread on utilise l'opération « ***stop*** » :

```
public final void stop();
```

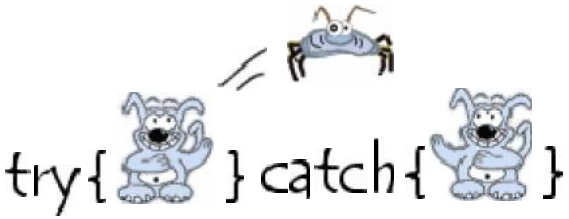
JAVA:POO

Les threads :synchronized pour gérer la concurrence d'accès

```
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();
}}
```

JAVA:POO

Les exceptions




```
try {
operation_risquée1;
opération_risquée2;
}
catch (ExceptionInteressante e) {
traitements
}
catch (ExceptionParticulière e) {
traitements
}
catch (Exception e) {
traitements
}
 finally {
traitement_pour_terminer_proprement;
}
```

JAVA:POO

Les exceptions

Si par contre on désire que l'exception soit traité par les blocs de niveaux supérieurs, il suffit d'inclure à la fin de la série d'instructions contenues dans le bloc *catch{}* une clause *throw*, suivie du type de l'exception entre parenthèse puis du nom de l'exception

```
class Nom_de_la_classe {
public static void main(String[] args)
 { // Instructions inoffensives (affectations, ...);
 try {
  // Instructions susceptibles de provoquer des erreurs;
  }
catch (TypeException e)
 {
 // Instructions de traitement de l'erreur;
    throw (TypeException)e; }
// Instructions si aucune erreur est apparue;
 } }
```