Cours Java

2011-12

Plan du cours

- 8 journées complètes.
- Thèmes abordés :
  - Retour sur algorithmique
  - UML (Unified Modeling Language)
  - Java et POO
  - Interfaces graphiques avec Swing.
  - Java et les bases de données
  - Java et le web : JSP, Servlet.
  - Projet type e-commerce

Bibliographie

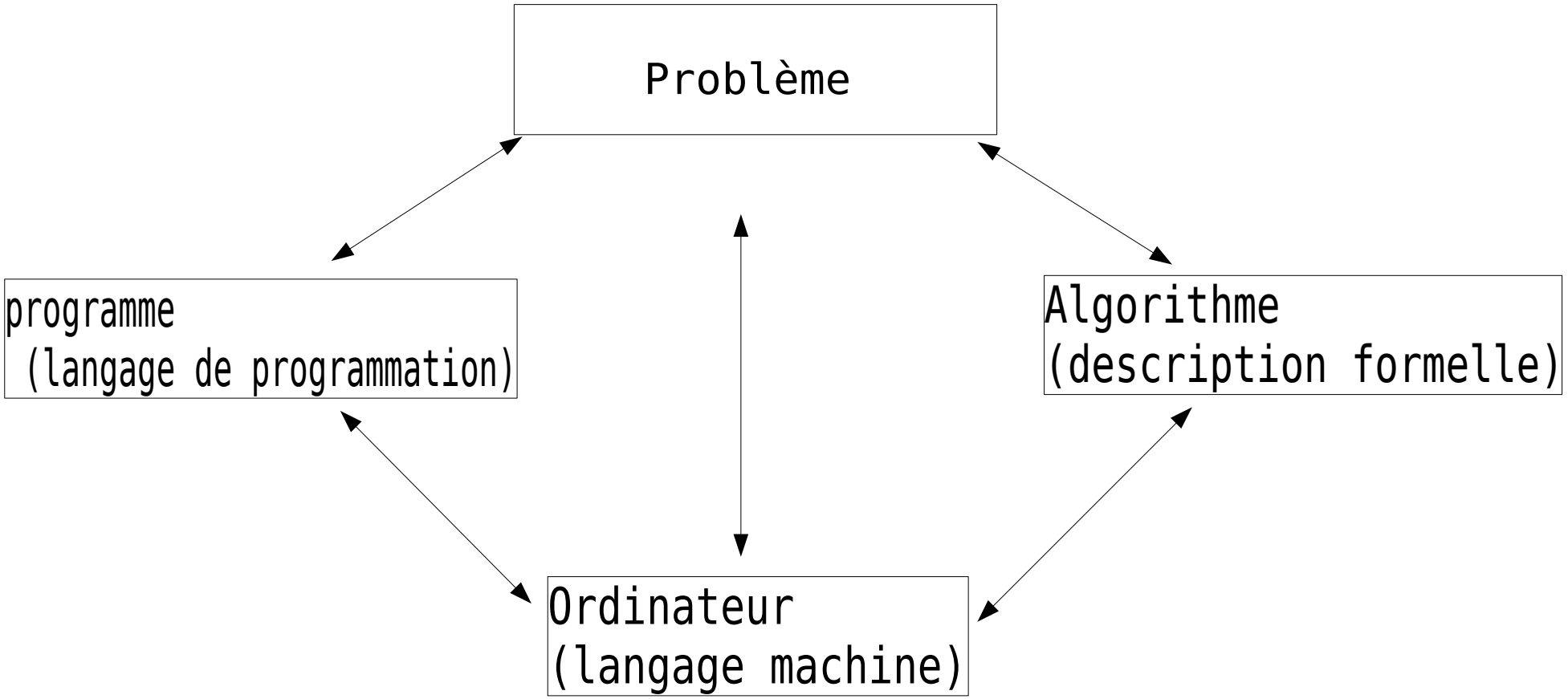
- Java in a nutshell 5ème édition française, O'reilly.
- Thinking in Java 2nde édition française (disponible gratuitement aux formats HTML et PDF sur le web : http://penserenjava.free.fr/).
- Le site java de Sun :
  - http://java.sun.com
  - et particulièrement le lien sur l'API : http://java.sun.com/j2se/1.4.1/docs/api/

0. Un petit retour sur l'algorithmique

Algorithme – Programme informatique

- Un algorithme :
  - est un processus de calcul permettant d'arriver à un résultat final déterminé
  - est un ensemble de règles opératoires dont l'application permet de résoudre un problème donné au moyen d'un nombre fini d'opérations.
    - nombre fini d'étapes, nombre fini d'opérations (effectives et sans ambiguïté) par étape, fournir un résultat.
- Un programme informatique :
  - la traduction d'un algorithme dans un langage de programmation.

Résolution informatique de problèmes



Les langages de programmation

- On classe souvent les langages de programmation en " générations " :
  - 1ère génération : langages machine (instructions sont représentées en binaire).
  - 2ème génération : langages d'assemblages (représentation symbolique)
  - 3ème génération : langages évolués (procéduraux, objets, fonctionnels et logiques).

Les langages évolués

- Langages impératifs offrent une certaine richesse des expressions. (exple : Pascal, C, Fortran, Cobol, Basic). Le programmeur explicite la suite d'actions que devra suivre l'ordinateur.
- Langages par objets sont une extension de la programmation impérative, elle permet l'association de code et données dans un "objet" (exple : Smalltalk, C++, Java).
- Langages déclaratifs :
  - Langages fonctionnels : se fondent sur la notion de fonction et sur la récurrence pour la définition implicite de fonction (exple : Lisp, Haskel)
  - Langages logiques : offrent à l'ordinateur, non pas un algorithme, mais des informations sur les données et les relations qui les lient aux résultats (exple: Prolog).

Interprétation / Compilation

- Les langages interprétés sont des langages décodés et exécutés instruction par instruction à l'aide d'un programme appelé **interpréteur**.
- Les langages compilés sont des langages où toutes les instructions sont traduites en code objet avant d'être exécutées, c'est le rôle du **compilateur**. Les phases sont les analyses lexicale, syntaxique et sémantique puis une production du programme (affectation d'adresses, optimisations, etc.).

Variable/Constante

- Un ordinateur est construit autour d'une mémoire divisée en cases dans lesquelles, on peut ranger des informations. Il est possible de consulter et modifier le contenu des cases. La notion de case se trouve dans la notion de **variable** d'un langage de programmation.
- Une **constante** est un concept identique à une variable mais sa valeur est affectée au début du programme et ne peut être modifiée au cours de l'exécution du programme.
- Constantes et variables possèdent souvent un type.

Variable et type

- Un type est défini par :
  - un ensemble de valeurs possibles.
  - un ensemble d'opérateurs manipulant de ces valeurs.
- Un langage fortement typé exige la déclaration explicite des types des objets avant leur usage (exple : C, Pascal, Java, etc.)
- Un langage est faiblement typé lorsque le contenu d'un objet détermine son type. Les vérifications de compatibilité de types se font lors de l'exécution (exple : Lisp, Prolog, php).

Principaux types

- boolean, valeurs possibles : true ou false
- int (entier)
- double (réel)
- char (un caractère)
- String (chaîne de caractères)

Instructions d'entrée/sortie

- Opération permettant d'afficher à l'écran (en language Java) :

```
System.out.println(" Bonjour ")
System.out.println(" Bonjour "+prenom)
```

- Opération permettant de saisir une valeur depuis le clavier (pour le moment en pseudo-codé):

```
saisir(prenom)
saisir(note)
```

Notion de bloc

- Dans un programme informatique, un bloc délimite un ensemble d'instructions.
- Syntaxe adoptée : {, pour début un bloc et } pour le fermer.
- Un bloc ouvert doit être fermé.
- Les instructions d'un programme se trouvent dans un bloc.
- On peut imbriquer les blocs.

Premiers programmes

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

```
public static void main(String args[])
{
 String prenom;
 saisir(prenom);
 System.out.println(" bonjour " + prenom );
}
```

Instruction conditionnelle

```
if (condition)
 blocInstruction1;
else blocInstruction2;
```

- Sémantique : si la condition est vraie, l'instruction conditionnelle aiguille sur le bloc d'instructions 1, et si elle est fausse, sur le bloc d'instructions 2.
- Les instructions conditionnelles peuvent être imbriquées les unes dans les autres.

Opérateurs de comparaison

- Les opérateurs de comparaison sont :
  - == (égalité)
  - != (différence)
  - <
  - <= (inférieur ou égal)
  - >
  - >= (supérieur ou égal)

Un autre exemple avec conditionnelle

```
public static void main(String args[])
{
 int note;
 saisir(note) ;
 if (note >=10)
  { System.out.println(" Accepté ") ; }
 else {
    if (note>=8)
     { System.out.println(" Oral ") ; }
    else { System.out.println(" Refusé ") ; }
    }
}
```

Exemple avec conditionnelle

```
public static void main(String args[])()
{
 float note ;
 saisir(note) ;
 if (note >= 10)
   System.out.println(" Accepté ") ;
 else
   System.out.println(" Refusé ") ;
}
```

Opérateurs logiques et arithmétiques

- Les opérateurs mathématiques supportés sont : +, -, / (division), * (multiplication) et % (reste de la division entière - modulo).
- Affectation où la variable a prend la valeur de b :a=b ou bien a¬ b.
- ET logique : && Exemple : if(a==1 && b<5)
- OU logique : || Exemple :if (a==5 || b ==6)
- Inverse : ! Exemple : if (!a)

Instruction itérative : while

```
while (condition)
    blocInstruction
```

- Sémantique : tant que la condition est vraie, on passe dans le bloc d'instructions. Dès que la condition est fausse, on passe à l'instruction suivante.
- On passe 0 à n fois dans le bloc d'instructions.

Instruction itérative : do while

```
do
   blocInstruction
while (condition) ;
```

- Sémantique : On passe dans le bloc d'instructions une première fois puis on va évaluer la condition. Tant que la condition est vraie, on passe dans le bloc.
- On passe 1 à n fois dans le bloc d'instructions.

Instruction itérative : for

```
for (initialisaiton;condition; incrémentation)
   blocInstruction
```

- Sémantique : On initialise une variable et tant que cette variable valide la condition (vraie) alors on passe dans le bloc d'instructions. Après la dernière instruction du bloc, on incrémente la variable et on va évaluer la condition à nouveau.
- On passe 0 à n fois dans le bloc d'instructions.

Approche modulaire

- L'écriture d'application informatique doit être modulaire. Les avantages sont :
  - éviter de trouver des blocs d'instructions redondants.
  - clarté des programmes, ceux-ci étant découpés en sous-programmes concis, limités à une tâche précise.
  - facilité de conception et de mise au point.

Sous-programme

- Un sous-programme (ou module) est une portion de programme formant un tout homogène, destiné à remplir une certaine tâche bien délimitée.
- Il peut être appelé à partir d'un programme principal, d'un autre sous-programme ou bien par lui-même (on parle alors de récursivité).
- Un sous-programme est paramétré et peut correspondre à une fonction ou une procédure.

Fonction / procédure

- Une fonction est un sous-programme retournant un résultat. L'appel est donc de la forme :

```
variable = nomFonction();
```

- Une procédure est un sous programme ne retournant aucun résultat. Appel de la forme :

```
nomProcédure();
```

Arguments et sous-programme

- On peut passer des paramètres aux sous-programmes. Ils se trouvent alors entre les parenthèses.
- On distingue 2 types de passage de paramètres:
  - par valeur, les paramètres du sous-programme appelant restent inchangés.
  - par adresse, les paramètres du sous-programme appelant peuvent être modifiés par le sous-programmme appelés.

Exemple de sous-programme (Syntaxe pseudo-codé)

```
procVal (entier val)
{
 val = val * 2
}
principal()
{
 entier val = 5
 procVal(val)
 afficher(val)
}
```

```
procAdr (entier adr val)
{
 val = val * 2
}
principal()
{
 entier val = 5
 procAdr(adr val)
 afficher(val)
}
```

Tableau

- De nombreuses applications font appel à la manipulation de données multiples ayant des caractéristiques communes. Dans de telles situations, on range les informations dans un **tableau**.
- Les données élémentaires partagent alors le même nom et être toutes du même type. Chaque élément est identifié par la mention du nom du tableau, suivie d'un ou plusieurs indices, compris chacun dans une paire de crochets.

Tableau (2)

- `Déclaration : int tab[10]`

correspond à la déclaration d'un tableau de 10 valeurs entières.

- Pour afficher le contenu de la 4ème cellule du tableau : afficher(tab[3])

Le premier indice d'une tableau est 0.

- Initialisation lors de la déclaration :

```
int tab[5] = {1,2,3,4,5};
```

Tableau (3) -Initialisation

```
public static void main(String args[])
{
 final int TAILLE=10;
 int tab1[TAILLE], cpt;
 for(cpt=0;cpt<TAILLE;cpt++)
  {
   tab1[cpt] = 0;
   System.out.println(" tab[" +cpt+ "]  = "+tab1[cpt]);
  }
}
```

Récursivité

- C'est la caractéristique des fonctions capables de s'appeler elles-mêmes de manière répétitive, jusqu'à ce que soit vérifiée une condition donnée.
- Ce procédé est utilisé pour mener à bien des calculs répétitifs dans lesquels chaque action est effectuée en fonction du résultat précédent.

Récursivité (2)

- Pour donner une solution récursive à un problème, 2 fonctions doivent être vérifiées :
  - le problème doit pouvoir être décrit sous forme récursive,
  - la résolution doit s'appuyer sur une condition d'arrêt.
- Exemple : Fonction factorielle
  - n! = 1*2*3*4* ...*n-1*n
  - n!=n*(n-1)! avec 1!=1

1. Le langage Java

Historique du langage Java

- Java ≠ Javascript
- A l'origine, le langage Oak développé par SUN au début des 90.
- En 1994, Oak change de nom pour devenir Java.
- Les deux premières versions, Java 1.0 (95) et Java 1.1 (97) sont orientées Web. Java 1.2 (98), renommé Java 2 devient généraliste.
- Java 2 regroupe 3 fois plus de paquetages que Java 1.0.

Caractéristiques du langage Java

Java est

- un langage orienté objet
- à classes (les objets sont décrits/regroupés dans des classes).
- dont la syntaxe est très proche de C.
- fourni avec le SDK (Software Development Kit) - outils de développement, paquetages.
- portable comme l'indique son slogan " write once, run everywhere ".

Premier programme Java

Le programme se nomme HelloWorld.java

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

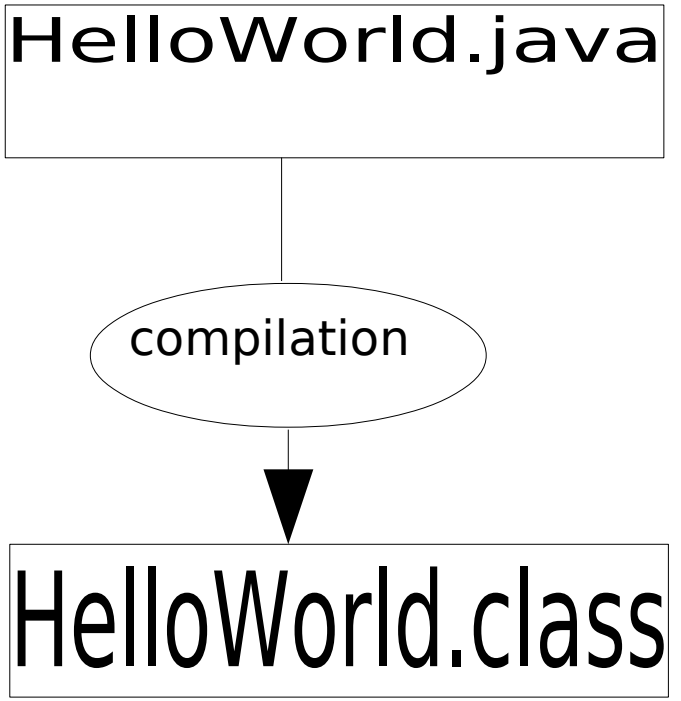
Compilation

Le code source ne peut être exécuté directement.
En java, le code source est traduit dans un langage spécifique à Java : le " bytecode ".
C'est le langage de la machine virtuelle java (JVM).
Ce langage est indépendant de la machine, donc grande portabilité, puisqu'il existe des JVM pour toutes les architectures.
Attention à la variable d'environnement CLASSPATH.

Compilation java (suite)

Notre programme source se nomme HelloWorld.java.

Après la compilation, on obtient un nouveau fichier HelloWorld.class, dont le contenu est du bytecode.



Ligne de commande :

```
javac HelloWorld.java
```

Exécution du bytecode

Le bytecode doit être exécuté par une JVM.

Cette machine virtuelle est simulée par un interpréteur, qui lit chaque instruction du programme (bytecode) et le traduit dans le langage du processeur de l'ordinateur et lance son exécution.

Ligne de commande :

```
java HelloWorld
```

Avantages et inconvénients de la JVM

Avantages :

- Portabilité, il existe des JVM pour l'ensemble des OS du marché.
- Le bytecode exporté est léger.
- Sécurité, la JVM lance de nombreuses vérifications sur le bytecode.

Inconvénients :

- Lenteur de l'exécution (compilation + interprétation) mais d'autres alternatives existent.

Quelques règles fondamentales

- Le nom d'une classe commence toujours par une majuscule.
- Les mots contenus dans un identificateur commencent par une majuscule : Hello**W**orld.
- Les constantes sont en majuscules.
- Les propriétés et les méthodes débutent par une minuscule.
- Ajouter des commentaires, la syntaxe est identique au C (/* ... */ ou //).

Eléments du langage
Retour sur le langage C(1)

- Les blocs : Ouverture et fermeture avec "{" et "}".
  - Arithmétiques : +,-,*,/,% (modulo)
  - Logiques : && (et logique), || (ou logique), ! (inverse).
  - Comparaisons : == (égalité), != (différence), >, <, <=, >=
  - Opérateurs unaires : ++, --
  - Assignations : +=, -=, *=, /=

```
a = a + 1;
équivalent à a++;
équivalent à a+=1
```

Eléments du langage
Retour sur le langage C(2)

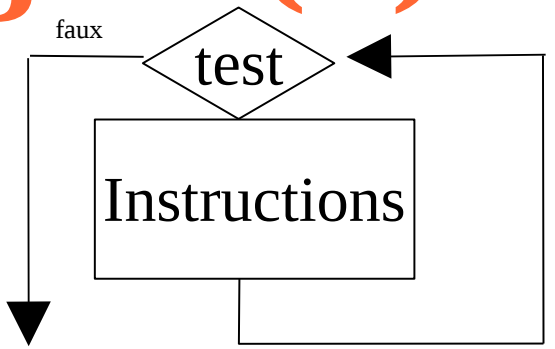
- Instruction conditionnelle :

```
if (condition)
  instruction1;
else
  Instruction2;
```

- Exemple Java:

```
if(a>=10)
  System.out.println(" Accepté ");
else
   System.out.println(" Refusé ");
```

Eléments du langage
Retour sur le langage C(3)



- La boucle while
- Syntaxe :

```
while(condition) {
    instructions
}
```

- Exemple :

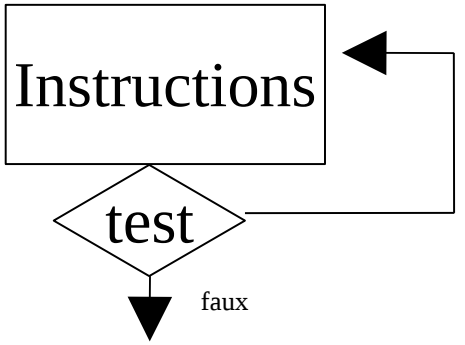
```
i = 0;
while(i<10)
{
  System.out.println("Valeur = "+ i);
    i++;
}
```

- Remarque : On passe 0 à n fois dans la boucle.

Eléments du langage
Retour sur le langage C(4)

- La boucle do while
- Syntaxe :

```
do
{
    instructions
} while(condition);
```



- Exemple :

```
i = 0;
do
{
    System.out.println("Valeur = "+ i);
    i++;
} while(i<10);
```

- Remarque : On passe 1 à n fois dans la boucle.

Eléments du langage
Retour sur le langage C(5)

- La boucle for
- Syntaxe :

```
for(initialisation;condition; pas)
{ instructions }
```

- Exemple :

```
for(i=0; i<10;i++) System.out.println("Val = " + i);
```

- Remarques : On passe 0 à n fois dans la boucle.
- Rédaction plus concise.

Manipulation d'une variable

- Deux techniques :
  - Par valeur où on utilise directement le nom de la variable.
  - Par adresse (ou référence) où la variable est rangée dans la mémoire.

Types primitifs

Déclaration des variables :

<type> <identificateur>;

Les types simples sont :

pour les entiers : byte (1 octet, [-128 à 127]), short (2 octets, [-32768,32767]), int (4 octets,[-2147483648, 2147483647]) et long (8 octets,[-9.10^{18},9.10^{18}]).

pour les réels : float (4 octets en simle précision) et double (8 octets en double précision).

le booléen: boolean (true ou false)

le type caractère : char (1 seul caractère).

Types non primitifs

Et les autres types :

- Les tableaux

```
int[] tab = new int[10];
```

- Les chaînes de caractères

```
String nom = "Bonjour le monde!";
```

- Les objets

```
class Personne {
   String nom;
   int age;
}
```

Variables et Java

- Les variables de type primitif (booléen, caractère, entier et réel) sont manipulées par valeur.
- Les variables de type non primitif sont manipulées par référence.

Exemple

```
..
int[] tab = {15,10,5,8};
System.out.println("Tableau = "+ tab);
..
```

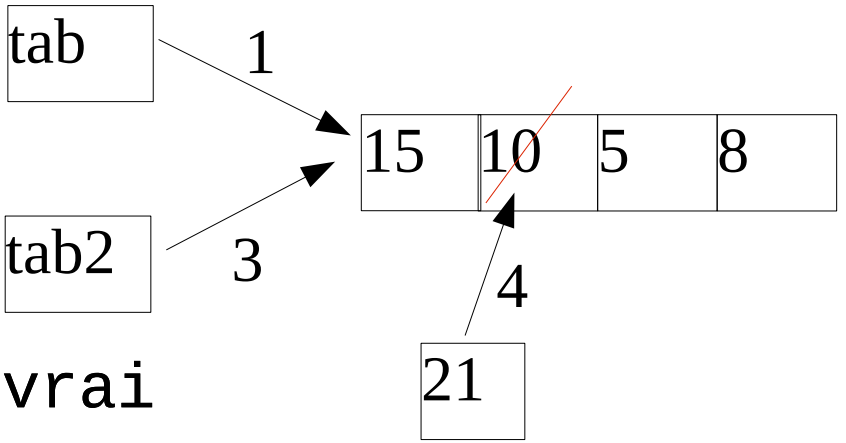
- Le résultat affiché est :

```
Tableau = [I@ad3ba4
```

l'adresse du tableau, explication : [pour un tableau, I pour integer et @xxxx pour l'adresse.

Exemple 2

```
..
1:int[] tab = {15,10,5,8};
2:int[] tab2 = new int[4];
3:tab2 = tab;
4:tab[1] = 21;
..
```




```
if(tab==tab2) // vrai
```

Exemple 3

```
..
1:int[] tab = {15,10,5,8};
2:int[] tab2 = new int[4];
3:for(int cpt=0; cpt<4;cpt++)
4:   tab2[cpt] = tab[cpt];
5:if(tab==tab2)
// faux, car 2 références distinctes.
```

2. La programmation orientée objet

Historique de la POO

Simula (1966) regroupe données et procédures.

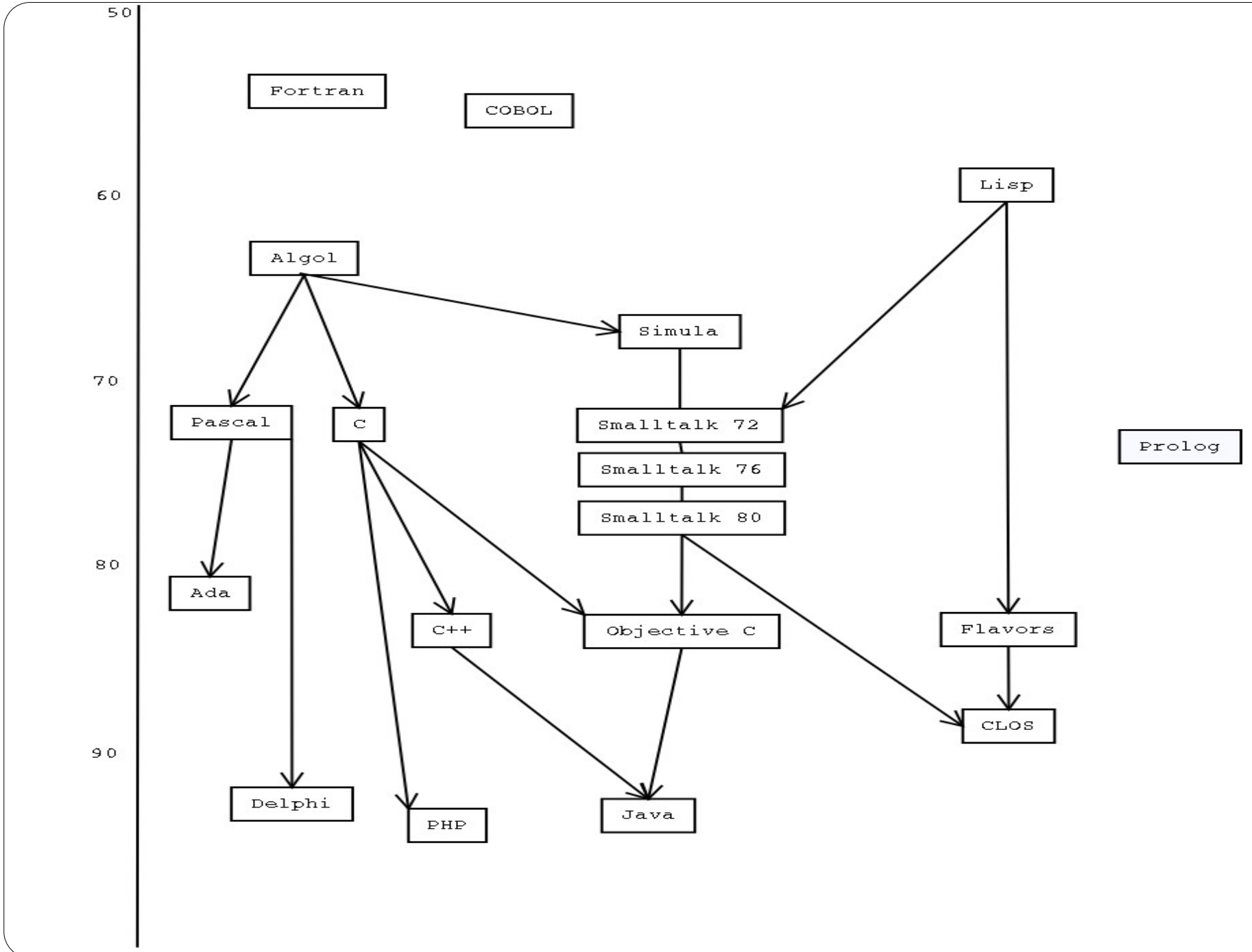
Simula I (1972) formalise les concepts d'objet et de classe. Un programme devient une collection d'objets actifs et autonomes.

Smalltalk (1972) : généralisation de la notion d'objet.

C++ (1983) : Extension du C s'inspirant de Simula.

Java (1994) : d'abord pour le web puis généraliste.

Autres : C#, eiffel, ada, clu, Object pascal, delphi, python, etc..


50
60
70
80
90
Fortran
COBOL
Lisp
Algol
Simula
Pascal
C
Smalltalk 72
Prolog
Smalltalk 76
Smalltalk 80
Ada
C++
Objective C
Flavors
CLOS
Delphi
PHP
Java

Motivation et avantages de POO

Motivation : concevoir, maintenir et exploiter facilement de gros logiciels.

Avantages : modularité, abstraction, productivité et réutilisabilité, sûreté, encapsulation.

Les autres approches : structurée impérative (C), fonctionnelle (Lisp) et logique (Prolog).

Modularité et Abstraction

Modularité : les objets forment des modules compacts regroupant des données et un ensemble d'opérations.

Abstraction : Les entités objets de la POO sont proches de celles du monde réel. Les concepts utilisés sont donc proches des abstractions familières que nous exploitons.

productivité et réutilisabilité, sûreté

productivité et réutilisabilité : Plus l'application est complexe et plus l'approche POO est intéressante en terme de productivité. Le niveau de réutilisabilité est supérieur à la programmation impérative.

sûreté : L'encapsulation et le typage des classes offrent une certaine robustesse aux applications.

Les concepts fondamentaux

- Objet, classe et instance
- les membres de classe et d'instance
- envoi de message et méthode
- héritage, encapsulation et polymorphisme
- Constructeur et destructeur
- Classe abstraite / concrète

Objet

Les objets sont omniprésents (humain, livre,etc.) dans notre monde.

En POO :

objet = identité + états + comportements

L'identité doit permettre d'identifier sans ambiguïté l'objet (adresse/référence ou nom).

Modélisation informatique : les états sont stockés dans des propriétés (variables) et les comportements sont implémentés à l'aide de méthodes (procédures / fonctions).

Exemple d'un objet

Soit la modélisation d'un objet être humain.

Son identité peut être son nom ou bien un numéro.

Ses états seront sa taille, son poids, la couleur de ses yeux, etc..

Ses comportements seront respirer, marcher, parler, etc..

Classe

Le monde réel regroupe des objets du même type.

Il est pratique de concevoir une maquette (un moule) d'un objet et de produire les objets à partir de cette maquette. En POO, une maquette se nomme une **classe**.

Une classe est donc un modèle de la structure statique (variables d'instance) et du comportement dynamique (les méthodes) des objets associés à cette classe.

Instance

Les objets associés à une classe se nomment des **instances**.

Une instance est un objet, occurrence d'une classe, qui possède la structure définie par la classe et sur lequel les opérations définies dans la classe peuvent être appliquées.

Membre

Membre = propriété + méthode

- Membre d'instance
- Propriété d'instance

  Une propriété d'instance est associée à une instance de la classe et non à la classe. Chaque objet possède donc sa propre copie de la propriété.
- Méthode d'instance

  Une méthode d'instance est associée à une instance d'une classe et non à la classe. Chaque objet possède donc sa propre copie de la méthode. Une méthode d'instance peut utiliser n'importe quel type de membre (classe ou instance).

Membre (suite)

- Membre de classe
- Propriété de classe

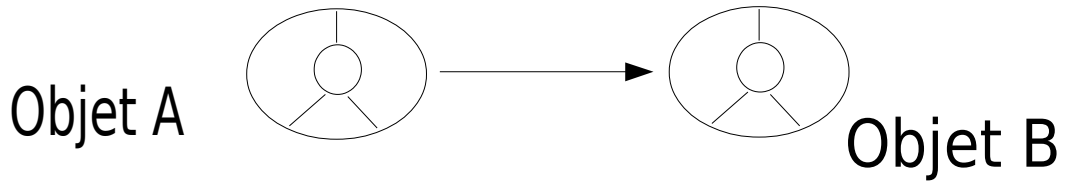
  Une propriété de classe est associée à sa classe et non à une instance de cette classe.
- Méthode de classe

  Une méthode de classe est associée à une classe et non à un objet.

  Une méthode de classe ne peut exploiter que des membres de classe.

Notion de message

Un objet seul ne permet pas de concevoir une application garantissant les objectifs de la POO.
Un programme est constitué d'objets. Ces derniers communiquent à l'aide de messages.
Un message est composé : du nom de l'objet recevant le message, du nom de la méthode à exécuter et des paramètres nécessaires à la méthode.

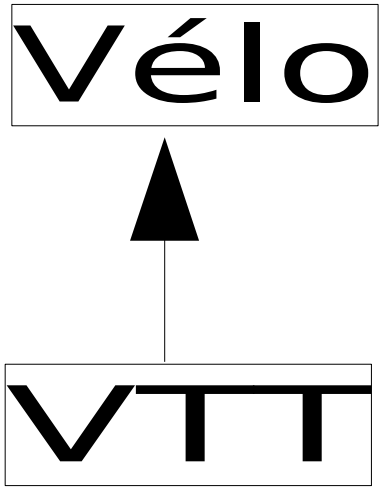


Héritage

Les objets sont définis à partir de classes. En POO, les classes sont définis à partir d'autres classes.

Par exemple : un VTT est une sous-classe de Vélo.

Une **sous-classe** n'est pas limitée aux caractéristiques de sa **super-classe**. Elle peut étendre cette dernière avec de nouvelles propriétés et méthodes.



Encapsulation

L'**encapsulation** est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet, c'est-à-dire en empêchant l'accès aux données par un autre moyen que les services proposés.

Ce mécanisme permet donc de garantir l'intégrité des données contenues dans l'objet.

Encapsulation (suite)

L'encapsulation permet de définir le niveau de visibilité des éléments de la classe. Ils définissent les droits d'accès :

- **publique**
- **privée**
- **protégée**.