Cours de Java

Ahmed Zidna, Bureau : B37.
Département Informatique de l'IUT
Univerisité de Lorraine, Ile du Saulcy, F-57045 METZ
ahmed.zidna@univ-lorraine.fr

Objectifs

- Programmer = Modéliser, structurer les données, concevoir des algorithmes, coder, tester.
- Acquérir les concepts de la programmation objet
- Acquérir les bases de la programmation objet Java
- Développer une application Java avec une interface

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Sommaire

Bibliographie

1 **JAVA La maîtrise**
Jérôme Bougeault
Le guide de formation Tsoft, Eyrolles

Bibliographie

1. **JAVA La maîtrise**
   Jérôme Bougeault
   Le guide de formation Tsoft, Eyrolles

2. **Références complètes Java**
   H.Schildt, First Interactive

Bibliographie

1. **JAVA La maîtrise**
   Jérôme Bougeault
   Le guide de formation Tsoft, Eyrolles

2. **Références complètes Java**
   H.Schildt, First Interactive

3. **Programmer en Java**
   Claude Delannoy, Eyrolles

Bibliographie

1. **JAVA La maîtrise**
   Jérôme Bougeault
   Le guide de formation Tsoft, Eyrolles

2. **Références complètes Java**
   H.Schildt, First Interactive

3. **Programmer en Java**
   Claude Delannoy, Eyrolles

4. **consulter des cours sur le web**

Java en quelques mots

Java n’est pas vraiment novateur :

- les fonctionnalités dont il dispose sont utilisées depuis des années par d’autres langages

Java en quelques mots

Java n’est pas vraiment novateur :

- les fonctionnalités dont il dispose sont utilisées depuis des années par d’autres langages
- Java en a fait une très bonne synthèse

Java en quelques mots

Java n’est pas vraiment novateur :

- les fonctionnalités dont il dispose sont utilisées depuis des années par d’autres langages
- Java en a fait une très bonne synthèse
- Java ne doit que sa syntaxe à C/C++

Java en quelques mots

Java n’est pas vraiment novateur :

- les fonctionnalités dont il dispose sont utilisées depuis des années par d’autres langages
- Java en a fait une très bonne synthèse
- Java ne doit que sa syntaxe à C/C++
- Java est orienté objet, on ne peut l’utiliser efficacement sans connaître ce type de programmation.

Introduction I

- Evolution des langages orienté objet :
  - **Simula (1967)** : un des ancêtre de Java, le premier à introduire les concepts de la POO.
  - **Smalltalk (1972)** : inspiré par Simula pour la POO et Lisp pour le côté language intérprété.
  - **C++ (1983)** : 1983 : Bjarne Soustrup a fait évoluer le langage C, de le faire évoluer au langage C++ qui offre les concepts de programmation orientée objet, le polymorphisme, ..etc.
  - **Java (1995)** : language plus simple, inspiré de Smalltalk et C++ (a éliminé par exemple : héritage multiple, surcharge des opérateurs..).
- Le langage **Java** est dit langage de programmation orientée objet (LOO) : dernière évolution du langage de programmation.

Evolution des langages I

langage machine $\Rightarrow$ assembleur $\Rightarrow$ langage haut niveau

$\approx$ 1950 — 1955 — FORTRAN 1957, PASCAL, COBOL, C

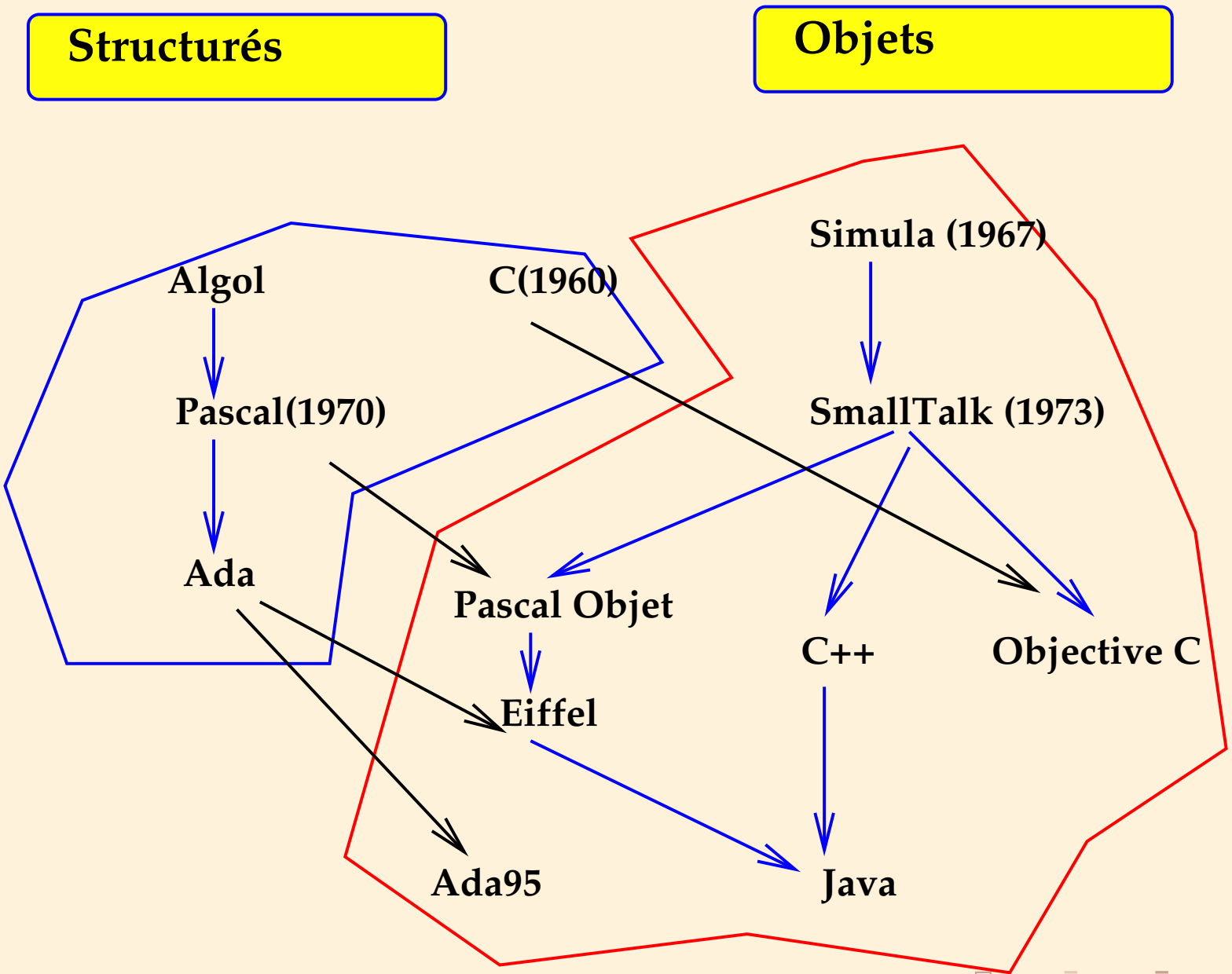
$\Rightarrow$ LOO

- Au cours de cette évolution, les langages s'éloignent des contraintes du matériel.
- Les LOO sont utilisés dans des domaines : *(CAO, IA, BdD, GL...)*.
- Ils sont adaptés à la réalisation de logiciel de grande taille.

Avantage de la POO I

- Ils apportent une plus grande souplesse de programmation :
- la possibilité de programmer de façon rigoureuse,
- la possibilité de contrôler très efficacement le code produit,
- un meilleur formalisme algébrique,
- la réutilisabilité du code.

Evolution du Langage



Java en quelques mots I

Pour SUN (Oracle), Java est un language :

- simple, orienté objet, distribué, interprété,
- robuste, sûr, indépendant de l'architecture, portable,
- efficace, multi-thread, dynamique,
- mais qui ne fait pas encore le café !

Simple .... mais pas simpliste I

Java est conçu pour être assimilé rapidement :

- faible nombre de constructions
- syntaxe familière au plus grand nombre (C, C++)
- évite les fonctionnalités rarement utilisées :
  - structures, unions,
  - héritage multiple, surcharge d'opérateur,
  - l'arithmétique sur les pointeurs, préprocesseur
- mémoire est gérée automatiquement

Orienté objet. Interprété I

1. **Orienté objet**
   - Java ne supporte que le style programmation objet contrairement à C++.
   - que des objets (des instances de classes) sauf les types de base tels que :int,double...
   - contrairement à C ou C++, le codage des types de base est indépendant de l'architecture
2. **Interprété**
   - le code source n'est pas traduit directement dans le langage de l'ordinateur
   - le compilateur génère des bytes codes indépendants de toute architecture.

Robuste. Efficace I

1. **Robuste**
   - fortement typé
   - gestion mémoire automatique (ramasse-miette)
   - gestion des exceptions beaucoup plus stricte qu'en C++
2. **Efficace**
   - Java est efficace par rapport aux langages interprétés tels que : Basic, perl, Tcl
   - actuelement, le byte code interprété est réputé 20 à 50 fois plus lent que C/C++

Distribué. Sûr I

1. **Distribué**
   - conçu pour supporter des applications réparties sur le réseau
   - il gère les protocoles TCP-IP. L'accès à une ressource grâce à une URL est proche de l'accès à un fichier.
2. **Sûr**
   - En Java, la sécurité est assurée par la robustesse du langage et par le runtime :
   - les bytes code sont typés, et on peut vérifier au chargement qu'il n'y a pas "tentative de fraude"
   - le client peut paramétrer les limitations du code (provenance, accès aux ressources)

Indépendance de l'architecture et Portabilité I

- les applications Java peuvent tourner sur n'importe quelle machine disposant de la machine virtuelle (JVM). Java sans recompilation.
- les accès aux ressources du système se font de façon homogène à travers une API ;
- l'utilisation de l'AWT permet une manipulation homogène de la GUI locale.
- le runtime est écrit en C ANSI, le compilateur en Java, ce qui rend les différentes architectures relativement faciles à supporter.

Multi-threading et Dynamique I

1. **Multi-threading :** Java prend en charge la programmation multiprocessus qui permet décrire plusieurs tâches simultanément.
   - le multi threading est supporté au niveau du langage (primitives de synchronisation, exclusion mutuelle)
   - le type de multi-threading utilisé n'est pas spécifié par le langage
2. **Dynamique**
   - car les classes sont chargées en cours d'exécution et .... uniquement lorsque c'est nécessaire

La doc API : Application Programming Interface I

La doc API : docs.oracle.com/javase/7/docs/api/ évolue tout le temps :

- **java.lang** : contient les classes les plus centrales du langage. Il contient la classe Object qui est la super-classe ultime de toutes les classes (String, Thread ...).
- **java.util** : complète java.lang(Date, Vector,..).
- **java.io** : contient les classes nécessaires aux entrées-sorties.
- **java.awt** : awt pour Abstract Window Toolkit contient des classes pour fabriquer des interfaces graphiques.

La doc API : Application Programming Interface I

- **java.applet** :utile pour faire des applets, applications utilisables à travers le Web.
- **java.sql** : pour l'accès aux base de données (JDBC : Java DataBase Connector).
- **java.rmi** : pour invoquer les méthodes d'objets distants (RMI : Remote Method Invocation).
- **java.net** : fournit une infrastructure pour la prog réseau.

Les outils Java I

- Les application fournies sont (entre autres)
- **javac** : le compilateur
- **java** : la machine virtuelle
- **javadoc** : le générateur de documentation
- **appletviewer** : la machine d'exécution d'applets
- **jar** : l'archiveur
- **jdb** : le débogueur.

Compiler et exécuter en Java ? I

- Tous les fichiers sources doivent avoir l'extension **.java**.

```
1  public class Somme{
     public static void main(String[] arg){
3    int i, somme = 0;
     for (i = 1; i <= 100; i++) somme += i;
5    System.out.println("la somme des 100"+"premiers entiers:"
         +somme);
     }
7    }
```

- Pour compiler : **javac Somme.java** $--> $ **Somme.class**
- Pour exécuter : **java Somme**

Compiler et exécuter en Java ? I

- Un programme Java est structuré en modules appelés classes
- Une classe est une suite de déclarations et de méthode

```
1  public class <nom> {
   public static void main (String[ ] args){
3  < corps de la mt'ethode main>
   }
5  }
```

```
1  public class PresenteToi{
   public static void main (String[ ] args){
3  System.out.println("Je␣m'appelle␣" + args[0] + "␣" +args
       [1]);
   }
5  }
```

Compiler et exécuter en Java ? I

- Voici le programme PresenteToi avec plus de contrôle d'erreur.

```
1  public class PresenteToi{
   public static void main (String[ ] args){
3  if (args.length ==0)
   System.out.println("je ne sais qui vous etes");
5  else{
   System.out.print("Vous etes");
7  for(int i=0;i<args.length;i++)
   System.out.print(" " + args[i]);
9  System.out.println("\n");
   }
11 }
   }
```

- On obtient les ext'ecutions suivantes :

```
Java PresenteToi David Tonsac Vous etes David Tonsac
```

Compilation Exécution