IJA - TD 2

Historique de Java

Ce qu'est Java, architecture, machine virtuelle

Caractéristiques du langage et principe de base

Le mot-clé static

Ant et le packaging d'applications

Java ?

- Printemps 90 : Naughton, Gosling et Sheridan :
  - *"Le consommateur est le centre du projet, il faut construire un environnement de petite taille avec une petite équipe et intégrer cet environnement dans une nouvelle génération de machines : des ordinateurs simples pour des gens normaux."*
- Printemps 91 : Microprocesseur grand public.
  - La "Green Team" prototype une machine de pilotage de l'électroménager
- Août 91 : Gosling développe Oak
- Août 92 : ----------duke---------->

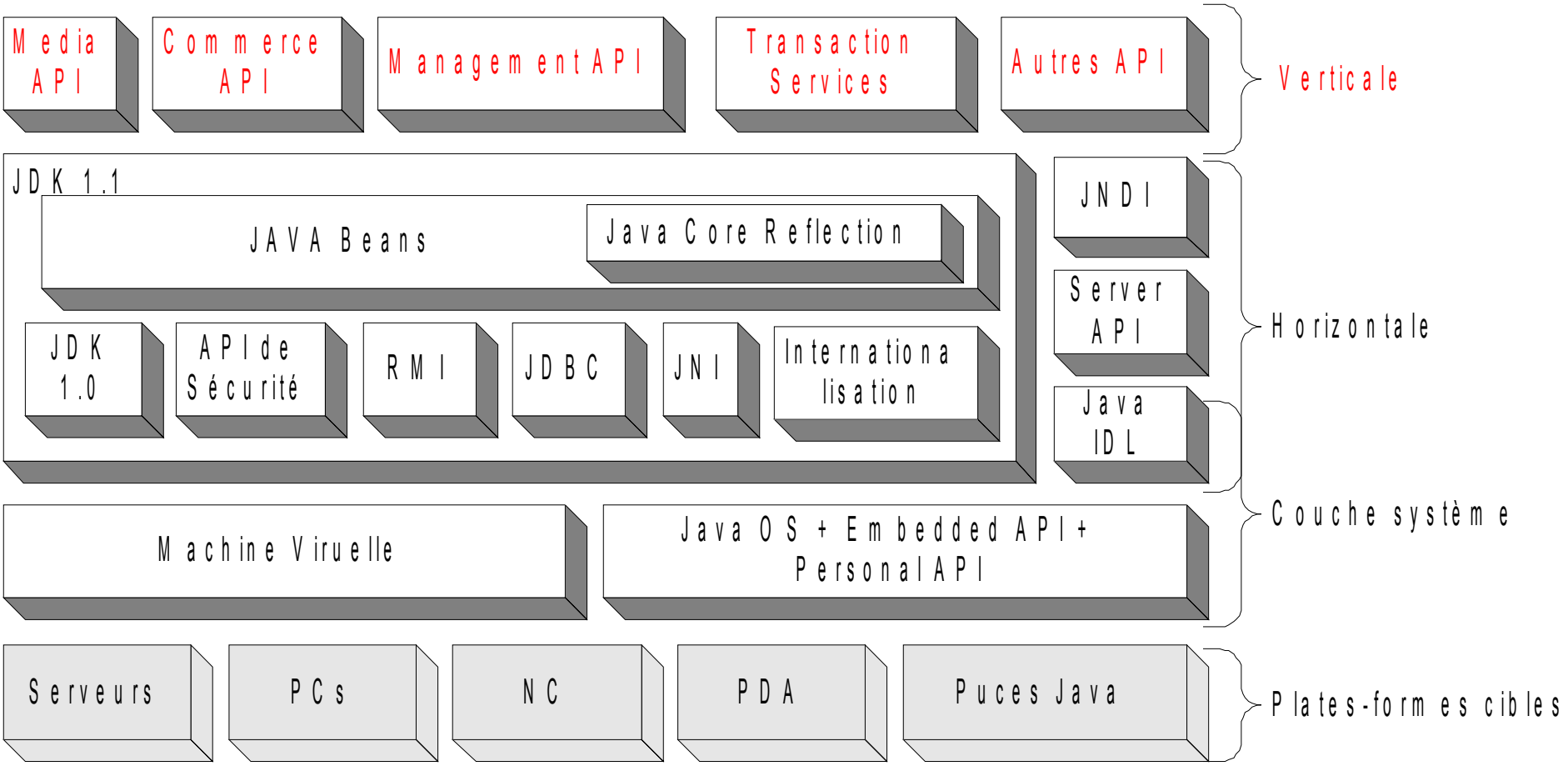
Java ?

- Eté 1993 : Sté. "First Person" est en train de couler
  - Eric Schmidt (Sun) demande une adaptation au Net
    - Gosling : travaille sur le code et ajoute les packages java.net
    - Naughton : cherche une application stratégique ==> hotjava
- Janvier 1995
  - Oak ==> Java, HotJava
- Août 1995 - Première licence sur Netscape
- Janvier 1996 - JDK 1.0.1
- Fév. 97 - JDK 1.1
- Jan 99 - JDK 2.0 (aka 1.2)
- 2003 J2se 1.4, J2ee, J2me (Standard, Entreprise, Micro) : 92Mo, 9200 classes,
- 2004 J2se 1.5, 230Mo, 13295 classes
- J2SDK, J2RE, J2EE

Java, ce que c'est !

- Une architecture technique
- Un langage OO
- Une bibliothèque de 5000 éléments

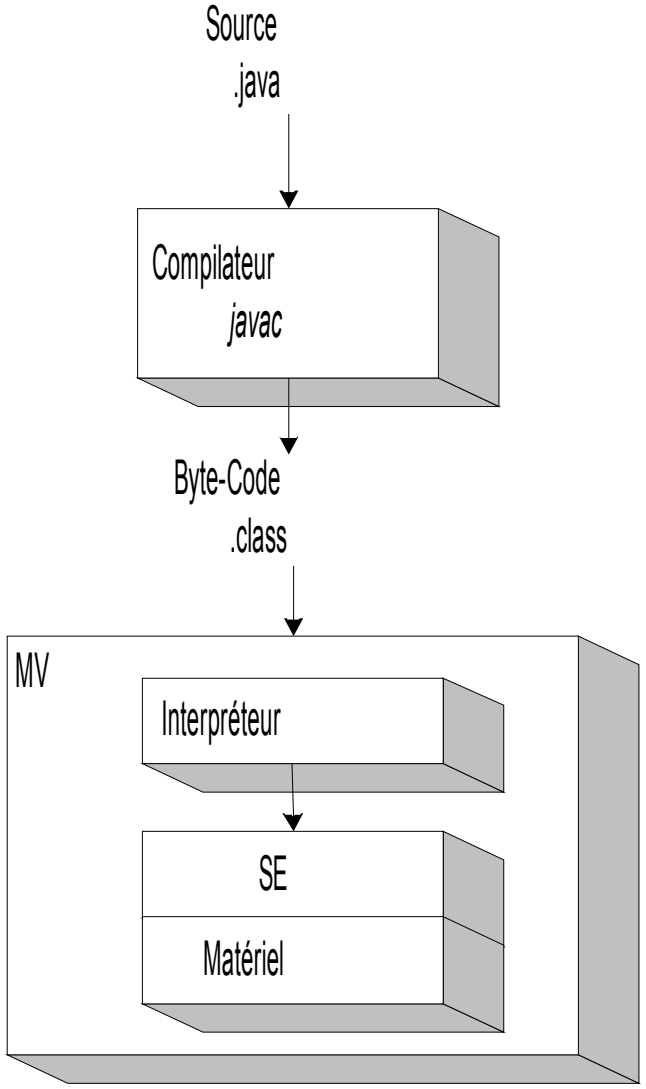
Plateforme Java : architecture



Java Client/Serveur : Nicolas, Avare, Najman - Eyrolles

Machine Virtuelle : Emulateur

- Byte-Code/P-Code/J-Code
  - opcode : 1 octet pour l'instruction
  - 0,n opérandes
- MicroProcesseur logiciel
  - Jeu d'instructions
  - Registres (pc, optop, frame, vars)
  - Pile, Heap
  - Ramasse-miettes
  - Espace de stockage des méthodes
  - Tas de constantes
- ==> Compilé ou Interprété



Caractéristiques du langage

- Portable --> Bytecode
- Sécurisé --> chargement sécurisé
- Auto-documenté --> javadoc
- Graphique --> swing, awt
- Distribué --> net

- Semi-reflexif --> forName
- Multi-thread --> thread

“l’œuf ou la poule?”

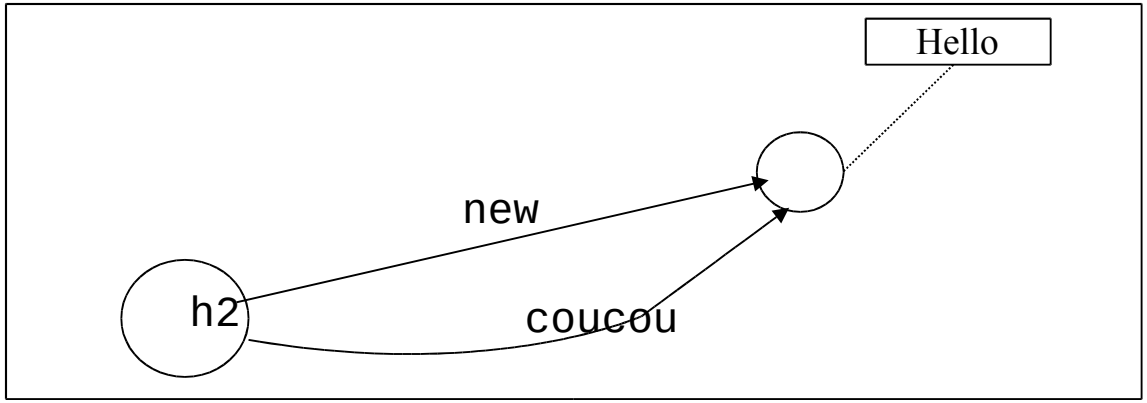
- Chaque objet provient d'une classe dont on fabrique une instance

```
Object o1=new Object();
```

- Mais comme “tout est objet”, comment fait-on pour avoir le premier objet ?
  - Soit il existe un premier objet fourni au démarrage de la machine virtuelle au démarrage
  - Soit il existe des objets toujours disponibles
  - ...

La classe vue comme un objet

- Toutes les classes instanciées sont connues de la machine virtuelle. Elles pourraient donc être également utilisées comme des objets.




```
...
Hello h2=new Hello();
Res r=h2.coucou();
...
```

```
class Hello{
  public void coucou(){}
}
```

Comment écrire une méthode coucou dans la classe hello, mais dont le destinataire ne soit pas une des instances, mais la classe ?

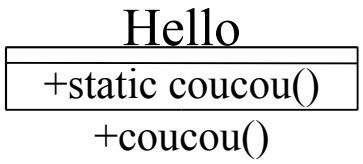
Le mot clé static

- On ajoute un mot-clé à la signature des méthodes

```
public void coucou(){ ... } /*Cette méthode est connue des instances-
   objets*/

public static coucou(){...} /*Cette méthode est connue de la classe */
```

- Comment l'objet h2 fait-il pour invoquer une méthode statique sur la classe Hello?
  - Soit on change d'opérateur : h2*coucou() ==> Indique qu'on invoque la méthode statique (mais c'est pas comme ça qu'on fait)
  - Soit on adapte le destinataire du message : Hello.coucou() ==> Indique que coucou est relatif à la classe et non pas à une de ses instances




```
public class Hello{
  public void coucou(){...}
  public void static coucou(){...}
}
```

```
...
Hello h2=new Hello();
h2.coucou();
Hello.coucou();
...
```

Exercice

- Ecrire le code d'une classe Client, utilisant une classe DivisionO réalisée en utilisant des méthodes d'objets
  - Compiler
- Ecrire une classe DivisionO en objet
  - Compiler, Lancer
- Modifier le code de la classe Client pour utiliser une classe DivisionS qui présente des méthodes statiques
  - Compiler
- Ecrire une classe DivisionS en statique
  - Compiler

Conséquences directes

- La classe est également vue comme un objet
- Les attributs et les méthodes statiques sont associés à la classe et non à chaque objet

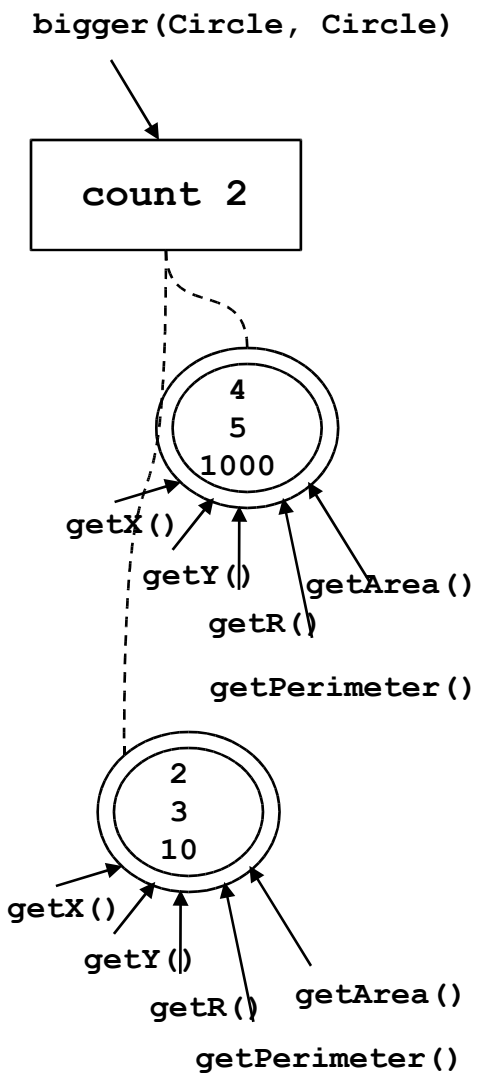
- Toutes les instances de la classe peuvent être vus comme partageant la valeur des attributs "static"
- Il n'est pas nécessaire d'instancier une classe pour accéder à ses éléments statiques

- La notion de static peut être perçue comme représentant les valeurs communes à toutes les instances d'une même classe...

Exemple du mot clé static

```
public class Circle {
   public static int count = 0;
   public static double PI = 3.14;
   protected double x, y, r;
   public Circle(double r){this.r = r; count++;}
    ... /*cf ppts précédents*/
   public static Circle bigger(Circle c1, Circle c2){
    if (c1.r > c2.r) return c1; else return c2;
   }
}

public class UneClasseExemple{
  public UneClasseExemple(){
    Circle c1 = new Circle(10);
    Circle c2 = new Circle(20);
    int n = Circle.count;                    // n = 2
    Circle c4 = Circle.bigger(c1, c2); // c4 = c2
   }
}
```

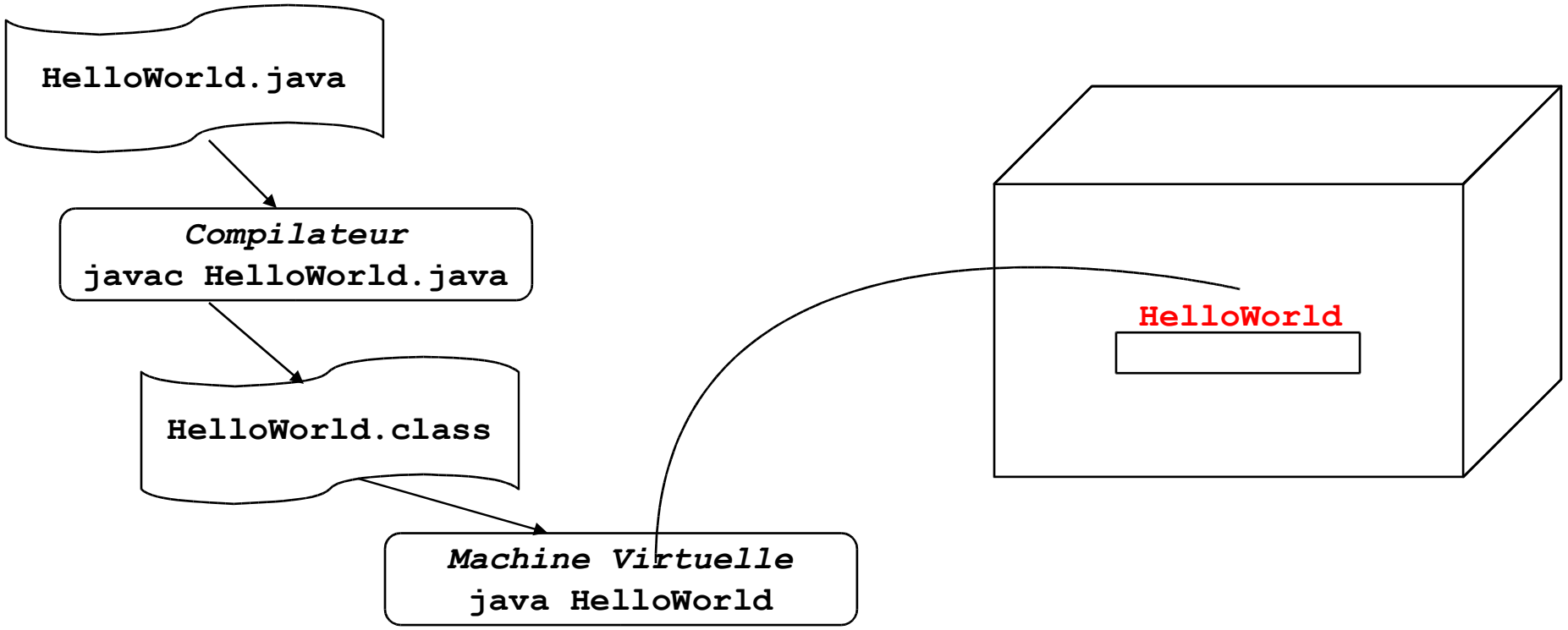


Enfin mon premier programme

```
public class HelloWorld {
  public static void main(String [] arg){
    System.out.println("Hello, World !");
  }
}
```

```
Static main(String [] arg)
```

HelloWorld



Exercice

- Décrire en terme d'objets et de messages la commande d'affichage

```
System.out.println("coucou");
```

- Lancer le programme de gestion des divisions.
  - Quelles sont les différences entre approche statique et l'autre ?
  - A quoi ressemble l'approche statique ?
- Réaliser une usine d'objets d'une classe Coucou
  - La classe FabriqueCoucou, permet d'instancier les objets de la classe Coucou. Réaliser le client, l'usine et la classe.

Pourquoi ?

- Pourquoi ne pas tout faire en statique ?
  - Parce qu'en programmation objet on manipule des objets qui sont beaucoup plus faciles à réorganiser
- Qu'est ce qui est statique ?
  - Le main (on n'a pas le choix, il n'y a pas encore d'objet)
  - Les constantes :

```
public static final int ETAT_STABLE=1;
….
    toto.setState(UneClasse.ETAT_STABLE);
```

  - Les fonctions communes de service
    - Dans ce cas on essaye de les regrouper dans une classe de service
  - Les usines de fabrication pour masquer les constructeurs

Le packaging d'applications

Le retour du makefile...

Objectifs du Makefile

- Gérer une chaîne complète de production de logiciels (et plus si affinité)
  - Ex :
    - Compilation
    - Génération de squelettes
    - Intégration de code
    - Tests
    - Packaging
    - Mailing
    - Déploiement...
  - ==> Le pb : chaque tâche ci-dessus repose sur des outils spécifiques. Il faut des outils pour mettre bout à bout l'exécution de ces outils
- Pour réaliser l'ensemble de ces tâches, il existe deux solutions :
  - Les langages de scripts
  - Les outils de gestion de projet comme Makefile ou Ant

Objectifs du Makefile

- Les objectifs du makefile sont de gérer proprement un ensemble de tâches. Ex en C :
  - compilation, linkage, exécution

```
run: test
    ./test
test: test.o
    gcc -o test.o test.c
test.o: test.c
    gcc -c test.c
```

- Les tâches déjà réalisées ne sont pas ré-exécutées, seuls les fichiers à regénérer le sont.
  - ant est une adaptation de make au monde java, et aux projets plus « complexes »

Ant

- Repose sur un fichier de description de tâches : build.xml
- La commande est `ant`
- Il existe toute une collection de tâches standardisées
  - java, javac, mail, cc, jar, tar, move, copy, documentation...
- Il est possible de définir de nouvelles tâches
- http://ant.apache.org

Ant exemple

```
<project name="jmxosgi" default="run" basedir=".">
  <property name="src" value="src"/>
  <property name="classes" value="classes"/>
  <property name="doc.dir" value="doc"/>

  <path id="classpath">
    <pathelement location="lib/osgi.jar"/>
  </path>

  <target name="clean">
    <delete verbose="false"
            includeEmptyDirs="true">
      <fileset dir="${classes}"/>
    </delete>
  </target>
  <target name="init">
    <mkdir dir="${classes}"/>
  </target>
```

```
  <target name="compile" depends="init">
    <javac srcdir="${src}"
           destdir="${classes}"
           debug="on">
      <classpath refid="classpath"/>
    </javac>
  </target>
  <target name="run" depends="compile">
    <java classname="Test">
     <classpath refid="${classpath}"/>
    </java>
  </target>
</project>
```

Les commandes possibles :

- ant
- ant init
- ant clean
- ant compile
- ant run

Organisation d'un projet en Java

- Un projet java est structuré en :

```
totor/          ==>   indique le nom du projet
   src/         ==>   contient les sources java à compiler
                      avec la structure en package (voir plus
                      tard...)
   classes/     ==>   contient les classes compilées
   doc/         ==>   contient la documentation du projet
   lib/         ==>   contient les librairies externes au
                      projet
   build.xml    ==>   Les directives de génération
```

- On peut également retrouver :
  - Un fichier de licence
  - Dans la documentation un quickStart/Howto

Les cibles ant

- Tout projet java doit au moins contenir les cibles principales suivantes:

```
clean      ==>Nettoie tous les fichiers générés par la
               chaîne
dist       ==>Fabrique une archive propre
               distribution du projet
<default>  ==>La cible par défaut fabrique de quoi
               lancer le projet.
```

- Exemple de cible dist :

```
<target name="dist" depends="clean">
  <jar jarfile="${release}" basedir=".." excludes="CVS">
        <include name="jmxosgi/lib/**"/>
        <include name="jmxosgi/${src}/**"/>
  </jar>
</target>
```

Exercice

- Réaliser votre projet Division sous ant.
- Votre outil doit vous permettre de :
  - compiler, lancer et packager votre projet

- A partir de maintenant tous vos projets de développement sont packagés dans des scripts ant...