Java Virtual Machine

- Le compilateur (javac) génère du code intermédiaire dans des fichiers .class à partir des sources .java
- La machine virtuelle java (java) interprète ce code intermédiaire qui est indépendant du système d'exploitation sur lequel est exécuté le programme

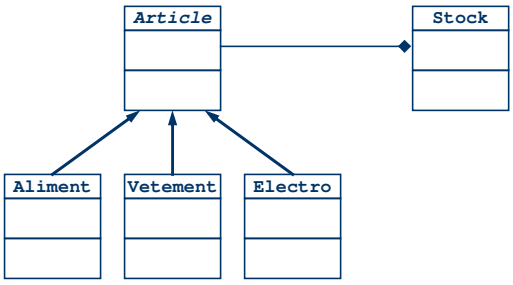
Java Virtual Machine

- Un fichier .class contient une table de symboles et des instructions (bytecode). Une décompilation est toujours possible. Elle permet de passer du fichier .class à la source .java. Des obfuscateurs rendent la source obtenue par décompilation difficile à exploiter.

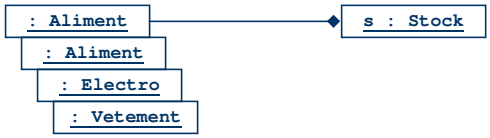
Java Virtual Machine…

- Le fichier .class est lu et chargé en mémoire sous forme d'un descripteur de classe (objet .class) par le chargeur de classes (ClassLoader).
- Pour chaque objet en mémoire à l'exécution d'un programme, il existe un objet qui décrit sa classe.

Avant compilation : des classes

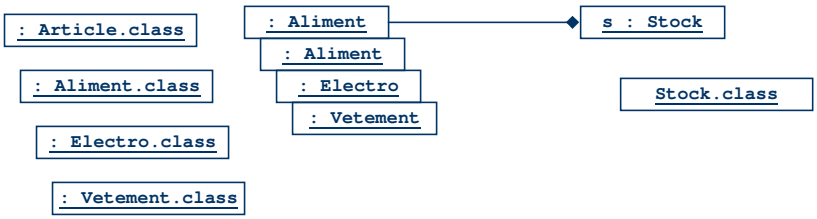


A l'exécution : des objets



A l'exécution : des objets …

dont un pour chaque classe chargée…



Chargement des classes

- Les classes sont chargées au fur et à mesure de l'exécution, en fonction des besoins (chargement paresseux – *lazy*).
- Le chargeur engendre des instances de **`java.lang.Class`** et maintient l'arbre d'héritage en mémoire (**`instanceof`**, **`super`** …).
- Ces instances de **`java.lang.Class`** sont des objets comme les autres, gérés par le ramasse-miettes.

La classe `Object`

```
+ getClass() : Class
+ equals(o : Object) : boolean
# clone() : Object
+ toString() : String
+ getName() : String
+ notify()
+ notifyAll()
+ wait()
+ wait(timeOut : long)
# finalize()
```

Chargement des classes…

- Le chargement d'une classe est implicite et tardif.

```
Exemple e; //pas de chargement
e = new Exemple(); //chargement si absente
```

- Il peut être explicite et immédiat.

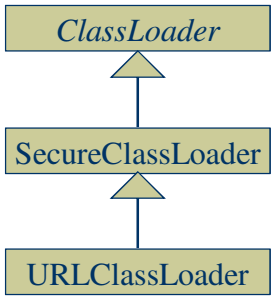
```
String nomCl = "Exemple";
//avec le chargeur par défaut
Class<?> c = Class.forName(nomCl);
//avec un chargeur spécifique
Class<?> c = Class.forName(nomCl, true, chargeur);
chargeur.loadClass(nomCl);
```

exécution du bloc static

La classe `Class`

```
+ toString() : String
+ forName(className : String) : Class
+ forName(className : String,
          init : boolean,
          loader : ClassLoader) : Class
+ newInstance() : Object
+ isInstance(obj : Object) : boolean
+ isInterface() : boolean
+ getName() : String
+ Constructor[] getConstructors()
+ Field[] getFields()
+ Method[] getMethods()
+ Class<?>[] getInterfaces()
+ …
```

Famille `ClassLoader`



ClassLoader sert de base pour la définition de chargeurs de classes.

SecureClassLoader permet le chargement d'une classe avec des restrictions d'accès.

URLClassLoader permet le chargement distant d'une classe depuis une archive ou un fichier .class à une URL.

Exemple avec `URLClassLoader`

```
public class Execute{
  public static void main(String[] args) throws Exception{
    URL urlJar  =
      new URL("http://jfod.cnam.fr/progAvancee/classes/utiles.jar");
    URL urlFolder =
      new URL("http://jfod.cnam.fr/progAvancee/classes/");

    // par défaut le classloader parent est celui de la JVM
    URLClassLoader classLoader =
      URLClassLoader.newInstance(new URL[]{urlJar,urlFolder});

    Class<?> classe = Class.forName(args[0], true, classLoader);
    Method m = classe.getMethod("main", new Class[]{String[].class});

    // recopie des paramètres,
    String[] paramètres = new String[args.length-1];
    System.arraycopy(args, 1, paramètres, 0, args.length-1);

    // exécution de la méthode main
    m.invoke(null, new Object[]{paramètres});
  }
}
```

Exemple avec `URLClassLoader`

- à essayer avec les commandes :

java Execute UneClasse un deux

Java Execute UneAutreClasse un deux