

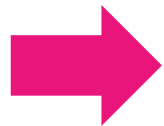
Outils de développement¹

Stéphanie CHOLLET

¹ *Software development tools that you can use to make your life easier* [J. Ferguson Smart, 2008]

En pratique...

- ▶ Le cycle de vie logiciel est un processus qui permet de produire un logiciel qui fonctionne.



Pas seulement coder !

Normalisation du code

Gestion de la documentation

Scripts pour la compilation,
le packaging et le déploiement
d'applications de manière reproductible

Gestion des versions
Version Control System

Suivi des problèmes
Bug tracker

Plate-forme de tests

Analyse statistique
du code

Et ...

Bien coder en Java

Objets et classes – 1/2

- ▶ Une **classe** est un modèle pour créer des objets qui ont des caractéristiques communes. La classe comporte la **structure d'un objet** (ses attributs et ses méthodes).
- ▶ En UML :

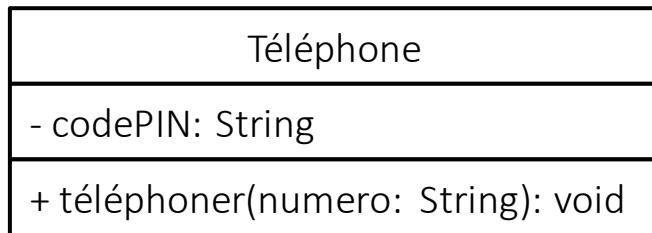


Diagramme de classes

```
public class Telephone {  
  
    private String codePIN;  
  
    public Telephone() {  
        super();  
    }  
  
    public Telephone(String codePIN) {  
        super();  
        this.codePIN = codePIN;  
    }  
  
    public void téléphoner(String numero) {  
        //Code pour téléphoner  
    }  
  
    public String getCodePIN() {  
        return codePIN;  
    }  
  
    public void setCodePIN(String codePIN) {  
        this.codePIN = codePIN;  
    }  
}
```

Objets et classes – 2/2

- Un **objet** possède des données (**attributs**) et des comportements (**méthodes**). Les objets appartiennent à une classe qui définit le type.

```
Telephone t1 = new Telephone("0000");  
System.out.println(t1.getCodePIN());  
t1.telephoner("0475750000");
```

```
Telephone t2 = new Telephone();  
t2.setCodePIN("1111");  
System.out.println(t2.getCodePIN());
```

- En UML :

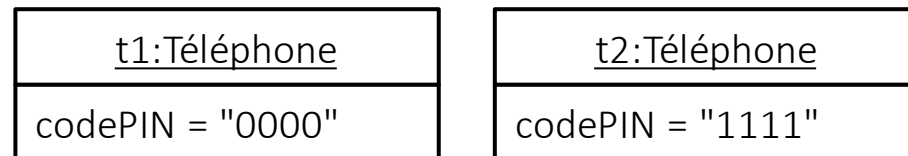


Diagramme d'objets

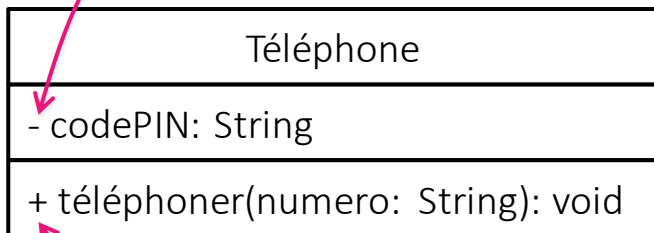


Un objet est une instance d'une classe

Remarques sur la modélisation d'une classe



Les constructeurs, les getters et les setters ne sont généralement pas représentés en Java



```
public class Telephone {
    private String codePIN;

    public Telephone() {
        super();
    }
    public Telephone(String codePIN) {
        super();
        this.codePIN = codePIN;
    }

    public void téléphoner(String numero) {
        //Code pour téléphoner
    }

    public String getCodePIN() {
        return codePIN;
    }
    public void setCodePIN(String codePIN) {
        this.codePIN = codePIN;
    }
}
```

Le ou les constructeurs

- ▶ Le rôle du constructeur est de permettre d'initialiser les données membre de la classe, ainsi que de permettre différentes actions (définies par le concepteur de la classe) lors de l'instanciation.
- ▶ Un constructeur :
 - ▶ Porte le même nom que la classe dans laquelle il est défini
 - ▶ N'a pas de type de retour (même pas *void*)
 - ▶ Peut avoir des arguments
 - ▶ Sa définition n'est pas obligatoire lorsqu'il n'est pas nécessaire :

```
public Telephone() {  
    super();  
}  
  
public Telephone(String codePIN) {  
    super();  
    this.codePIN = codePIN;  
}
```

Visibilités en Java – 1/2

- Pour les attributs et les méthodes :

Visibilité	En UML	En Java	Explications
Publique	+	public	Accessible partout où la classe est accessible, hérité par les sous-classes
Privée	-	private	Accessible que depuis la classe elle-même
Protégée	#	protected	Accessible depuis la classe elle-même et hérité par les sous-classes, accessible aussi par le code situé dans le même paquetage
Paquetage	~		Accessible que par le code situé dans le même paquetage et hérité que par les sous-classes du même paquetage



Rien

Visibilités en Java – 2/2

	Classe	Package	Sous-classe	Reste du monde
private	Oui	Non	Non	Non
paquetage	Oui	Oui	Non	Non
protected	Oui	Oui	Oui	Non
public	Oui	Oui	Oui	Oui

```
public class A {  
    private String a1;  
    String a2;  
    protected String a3;  
    public String a4;  
}
```

- ➡ Structurer correctement le code avec des paquetages
- ➡ Mauvaise spécification → ouverture de failles de sécurité
- ➡ Privilégier les attributs privés → accesseurs pertinents

Convention de nommage – 1/2

- ▶ Exemple : <https://www.oracle.com/technetwork/java/codeconventions-135099.html>

Identifier Type	Rules for Naming	Examples
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names. Subsequent components of the package name vary according to an organization's own internal naming convention.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese fr.esisar.cs440
Classes	Class names should be nouns , in mixed case with the first letter or each internal word capitalized .	class Raster class ImageSprite
Interfaces	Interface names should be capitalized like class names .	interface RasterDelegate interface Storing

Convention de nommage – 2/2

Identifier Type	Rules for Naming	Examples
Methods	Methods should be verbs , in mixed case with the first letter lowercase , with the first letter of each internal word capitalized.	<code>run();</code> <code>runFast();</code> <code>getBackground();</code>
Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter . Internal words start with capital letters.	<code>int i;</code> <code>char c;</code> <code>float myWidth;</code>
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by undercores .	<code>static final int MIN_WIDTH = 4;</code> <code>static final int MAX_WIDTH = 999;</code> <code>static final int GET_THE_CPU = 1;</code>

Quelques mots clés particuliers...

► static

- Définition unique quel que soit le nombre d'objets instanciés
- Permet de partager une variable de classe entre toutes les instances d'une même classe

```
public class Telephone {  
    private static int compteur = 0;  
    ...  
}
```

```
Telephone t1 = new Telephone("0000");  
Telephone t2 = new Telephone("1111");
```

```
System.out.println(t1.getCompteur()); 0  
System.out.println(t2.getCompteur()); 0  
t1.setCompteur(4);  
System.out.println(t1.getCompteur()); 4  
System.out.println(t2.getCompteur()); 4
```

► final

- Permet de rendre l'entité sur laquelle il s'applique non modifiable une fois qu'elle a été déclarée et initialisée

```
public class Telephone {  
    public final int nbTouches = 12;  
    ...  
}
```

► Définition d'une constante :

```
public static final int NB_TOUCHES = 12;
```

Quelques méthodes particulières... – 1/3

► Exécution du corps de l'application

```
public static void main(String[] args) {  
    for(int i = 0 ; i < args.length ; i++) {  
        System.out.print(args[i] + " ");  
    }  
    System.out.println();  
}
```

► La méthode toString()

```
@Override  
public String toString() {  
    return "Telephone [codePIN=" + codePIN + "]";  
}
```

► Les accesseurs et mutateurs : getter() et setter()

```
public String getCodePIN() {  
    return codePIN;  
}  
  
public void setCodePIN(String codePIN) {  
    this.codePIN = codePIN;  
}
```

Quelques méthodes particulières... – 2/3

► equals()

- Vérifie si 2 instances sont sémantiquement équivalentes même si ce sont 2 instances distinctes
- Chaque classe peut avoir sa propre implémentation de l'égalité mais généralement 2 objets sont égaux si tout ou partie de leurs états sont égaux

```
String chaine1 = new String("test");  
String chaine2 = new String("test");  
boolean isSame = (chaine1 == chaine2);  
System.out.println(isSame);  
boolean isEqual = (chaine1.equals(chaine2));  
System.out.println(isEqual);
```

false

true

► Pour rappel :

- L'opérateur == vérifie si 2 objets sont identiques : comparaison des références mémoire (sont-ils le même objet ?)

Quelques méthodes particulières... – 3/3

► hashCode()

- Retourne la valeur de hachage calculée sur l'instance d'un objet
- Cette valeur de hachage est essentiellement utilisée par les collections de type HashXXX (Hashtable, HashMap, HashSet...) afin d'améliorer leur performance

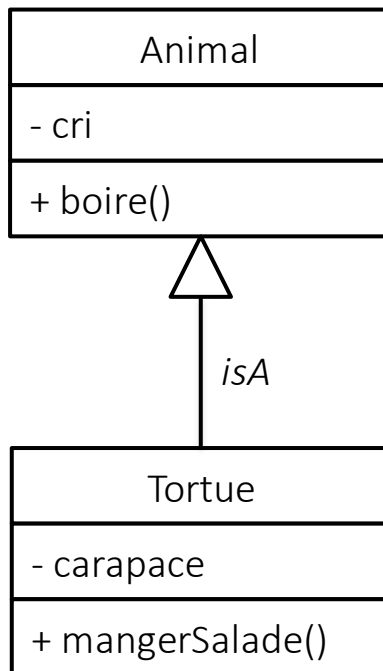
```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((codePIN == null) ? 0 : codePIN.hashCode());
    return result;
}
```

Java 1.7 et supérieur :

```
@Override
public int hashCode() {
    return Objects.hash(codePIN);
}
```

L'héritage

- L'héritage est un **lien** entre des classes. Si une classe est créée à partir d'une classe déjà existante, elle **hérite** de ses attributs et de ses méthodes.

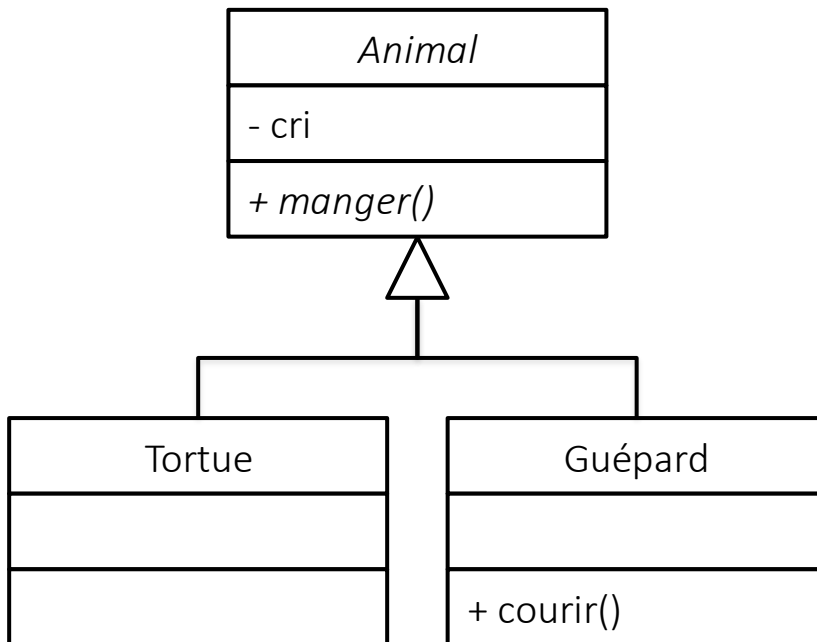


```
public class Animal {  
  
    private String cri;  
  
    public void boire() {  
        ...  
    }  
    ...  
}
```

```
public class Tortue extends Animal {  
  
    private Boolean carapace;  
  
    @Override  
    public void boire() {  
        // TODO Auto-generated method stub  
        super.boire();  
    }  
    public void mangerSalade() {  
        ...  
    }  
}
```


Classe abstraite – 1/2

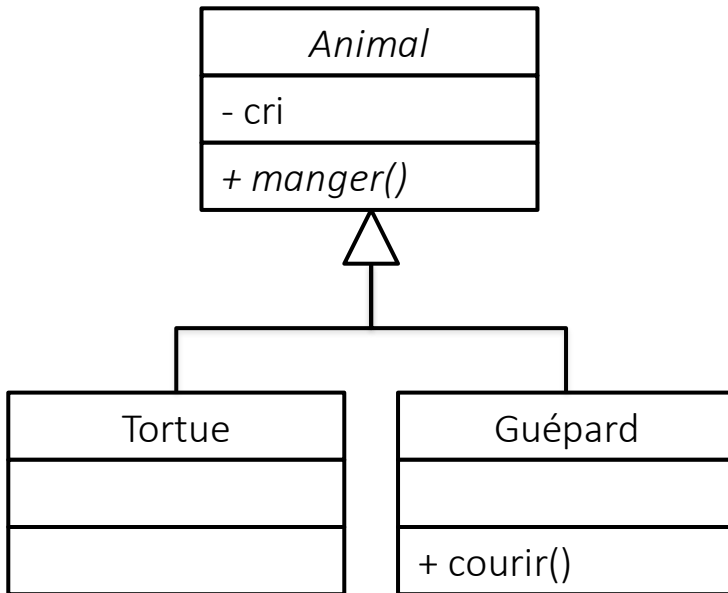
- Une **classe abstraite** est quasiment identique à une classe normale sauf qu'elle **n'est pas instanciable**. Elle contient **au moins une méthode abstraite**.



```
public abstract class Animal {  
  
    private String cri;  
  
    public Animal(String cri) {  
        this.cri = cri;  
    }  
  
    public abstract void manger();  
  
    public String getCri() {  
        return cri;  
    }  
}
```

- Le corps de la méthode abstraite est spécifié dans toutes les classes filles et il est **spécifique à chaque classe fille**.

Les classes abstraites – 2/2



```
public class Tortue extends Animal {

    public void manger() {
        System.out.println("Salade");
    }
}
```

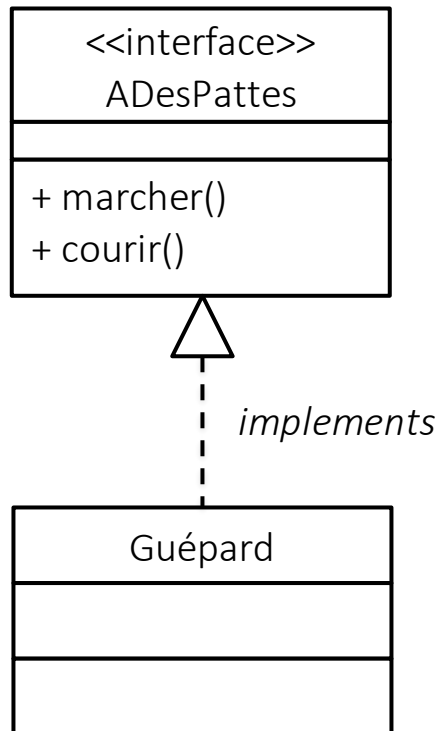
```
public class Guepard extends Animal {

    public void manger() {
        System.out.println("Viande");
    }
    public void courir() {
        System.out.println("Courir");
    }
}
```

```
Tortue tortue = new Tortue("wigu");
tortue.manger();
Guepard guepard = new Guepard("rrrr");
guepard.manger();
guepard.courir();
Animal animal = new Animal("cri");
Tortue tortue2 = new Animal("wiwi");
```

Les interfaces

- Les interfaces permettent de définir des méthodes devant être supportées par un objet **sans avoir à fournir l'implémentation** de ces méthodes



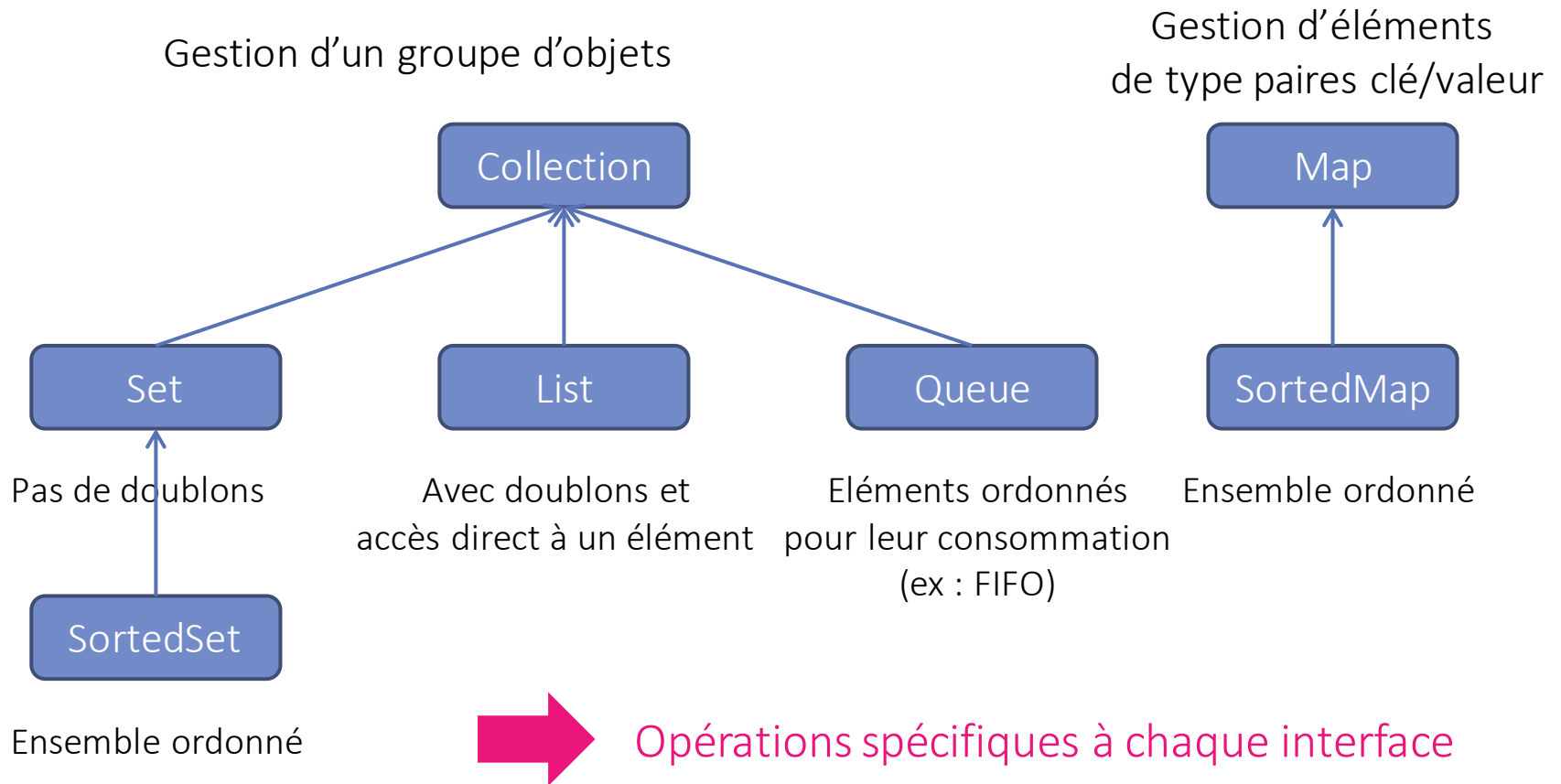
```
public interface ADesPattes {
    public void marcher();
    public void courir();
}
```

```
public class Guepard implements ADesPattes {
    public void marcher() {
        // Code
    }
    public void courir() {
        // Code
    }
}
```

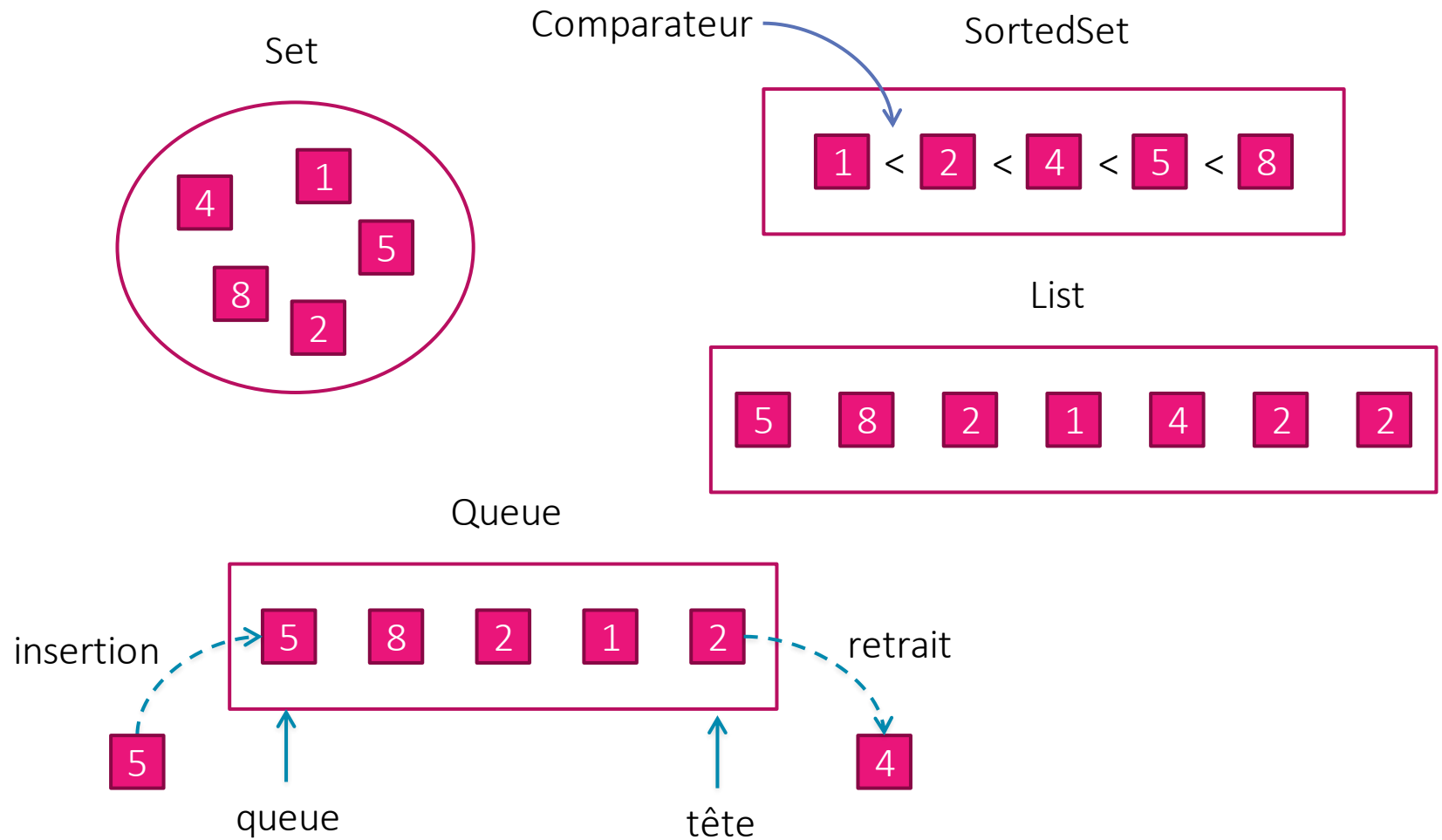
```
ADesPattes animal1 = new Guepard();
animal1.marcher();
Guepard guepard = new Guepard();
guepard.marcher();
ADesPattes animal2 = new ADesPattes();
```

Les collections en Java – 1/3

► Principales interfaces disponibles (java.util.XXX) :



Les collections en Java – 2/3



Les collections en Java – 3/3

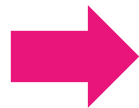
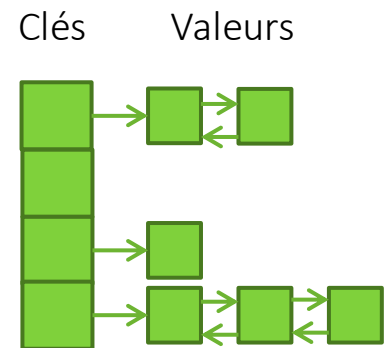
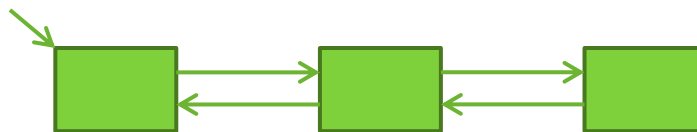
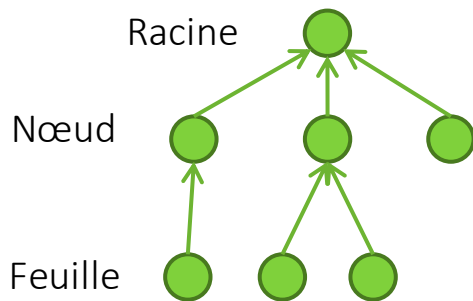
► Classes implémentant les interfaces :

		Utilisation générale	Gestion des accès concurrents
Interfaces	List	ArrayList LinkedList	Vector Stack CopyOnWriteArrayList
	Set	HashSet TreeSet LinkedHashSet	CopyOnWriteArrayList ConcurrentSkipListSet
	Map	HashMap TreeMap LinkedHashMap	Hashtable ConcurrentHashMap ConcurrentSkipListMap
	Queue	LinkedList ArrayDeque PriorityQueue	ConcurrentLinkedQueue LinkedBlockingQueue ArrayBlockingQueue PriorityBlockingQueue DelayQueue SynchronousQueue LinkedBlockingDeque

Les collections en Java – 4/3

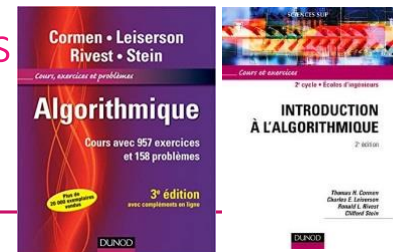
► Exemples de structures de données utilisées :

	Set	List	Map
Tableau redimensionnable		ArrayList, Vector	
Arbre	TreeSet		TreeMap
Liste chaînée		LinkedList	
Table de hachage	HashSet		HashMap, Hashtable



Complexité algorithmique des opérations

T. Cormen *et al.*, Introduction à l'algorithmique, 3^{ème} édition, Dunod





Eclipse est votre ami !

The screenshot shows the 'New Java Class' dialog box in Eclipse. The title bar says 'New Java Class'. Inside, the 'Java Class' section has a green 'C' icon and the text 'Create a new Java class.' Below this, there are three 'Browse...' buttons for 'Source folder:', 'Package:', and 'Enclosing type:'. The 'Source folder:' is set to 'Calculatrice-ex/src' and 'Package:' is set to 'fr.esisar.calculatrice'. The 'Name:' field is empty. Under 'Modifiers:', there are radio buttons for 'public' (selected), 'default', 'private', and 'protected', and checkboxes for 'abstract', 'final', and 'static'. The 'Superclass:' field is set to 'java.lang.Object'. The 'Interfaces:' section is empty with 'Add...' and 'Remove' buttons. At the bottom, there are checkboxes for 'Which method stubs would you like to create?' with options: 'public static void main(String[] args)', 'Constructors from superclass', and 'Inherited abstract methods' (checked). There is also a checkbox for 'Do you want to add comments?' with a link to 'Configure templates and default value here' and a 'Generate comments' checkbox. At the very bottom are 'Finish' and 'Cancel' buttons.

Répertoire dans lequel va être stocké la classe

Package de la classe

Nom de la classe

Caractéristiques de la classe (visibilité, abstraite...)

Nom de la super-classe pour cette classe

Ensemble des interfaces implantées par la classe

Méthodes dont le squelette sera implantée pour la classe

Génération des commentaires



Eclipse est vraiment votre ami !

Menu Item	Shortcut	Category
Undo Typing	Ctrl+Z	Commentaires
Revert File		
Save	Ctrl+S	
Open Declaration	F3	Indentation et formatage
Open Type Hierarchy	F4	
Open Call Hierarchy	Ctrl+Alt+H	
Show in Breadcrumb	Alt+Shift+B	Gestion des imports
Quick Outline	Ctrl+O	
Quick Type Hierarchy	Ctrl+T	
Open With		Génération de méthodes
Show In	Alt+Shift+W	
Cut	Ctrl+X	
Copy	Ctrl+C	Renommage
Copy Qualified Name		
Paste	Ctrl+V	
Raw Paste		
Quick Fix	Ctrl+1	
Source	Alt+Shift+S	
Refactor	Alt+Shift+T	

Menu Item	Shortcut
Toggle Comment	Ctrl+7
Remove Block Comment	Ctrl+Shift+\ Alt+Shift+J
Generate Element Comment	
Correct Indentation	Ctrl+I
Format	Ctrl+Shift+F
Format Element	
Add Import	Ctrl+Shift+M
Organize Imports	Ctrl+Shift+O
Sort Members...	
Clean Up...	
Override/Implement Methods...	
Generate Getters and Setters...	
Generate Delegate Methods...	
Generate hashCode() and equals()...	
Generate toString()...	
Generate Constructor using Fields...	
Generate Constructors from Superclass...	
Externalize Strings...	

←

Processus logiciel en Java

Du code à la livraison

Du code à l'exécution

```
public class Animal {  
    private String cri;  
    public void boire() {  
        ...  
    }  
    ...  
}
```

Compilation

Bytecode

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000130	00	48	00	49	01	00	18	2A	2A	2A	2A	2A	2A	2A	2A	2A	H.I.
00000140	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	2A	01
00000150	00	10	45	6E	74	65	72	20	75	73	65	72	6E	61	6D	65	.Enter username
00000160	3A	20	01	00	10	6A	61	76	61	2F	6C	61	6E	67	2F	4F	...java/lang/O
00000170	62	6A	65	63	74	07	00	40	0C	00	4A	00	4B	0C	00	4C	bject_@_J_R_I
00000180	00	4D	01	00	10	45	6E	74	65	72	20	70	61	73	73	77	M_ Enter passw
00000190	6F	72	64	3A	20	0C	00	4E	00	4F	01	00	19	2D	2D	2D	ord:..N.O.---
000001A0	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	2D	-----
000001B0	2D	2D	2D	2D	2D	2D	2D	0C	00	22	00	23	01	00	18	53	74
000001C0	61	74	75	73	3A	3A	4C	6F	67	69	6E	20	53	75	63	63	atus: Login Succ
000001D0	65	73	66	75	6C	6C	01	00	14	53	74	61	74	75	73	3A	esfull..Status:
000001E0	3A	4C	6F	67	69	6E	20	46	61	69	6C	65	64	01	00	0F	..Login Failed..
000001F0	72	71	21	21	54	68	61	6E	68	20	79	6F	75	21	21	01	!!!Thank you!!!

Exécution



Exécution



Pour compiler

► Prérequis :

- Compilateur javac disponible dans le JDK (*Java Development Toolkit*)

► Compilation :

- `javac Zoo.java Animal.java Guepard.java`



Génération de fichiers `.class` (bytecode)

Principe facile pour les TP mais ce n'est pas la réalité !



Le résultat de la compilation doit être dans un répertoire dédié (classes ou build/classes) et contenir l'arborescence initiale (= conserver les sous-répertoires) !

Pour exécuter

- ▶ Prérequis :

- ▶ Avoir un environnement d'exécution Java (*Java Runtime Environment*)

- ▶ Exécution :

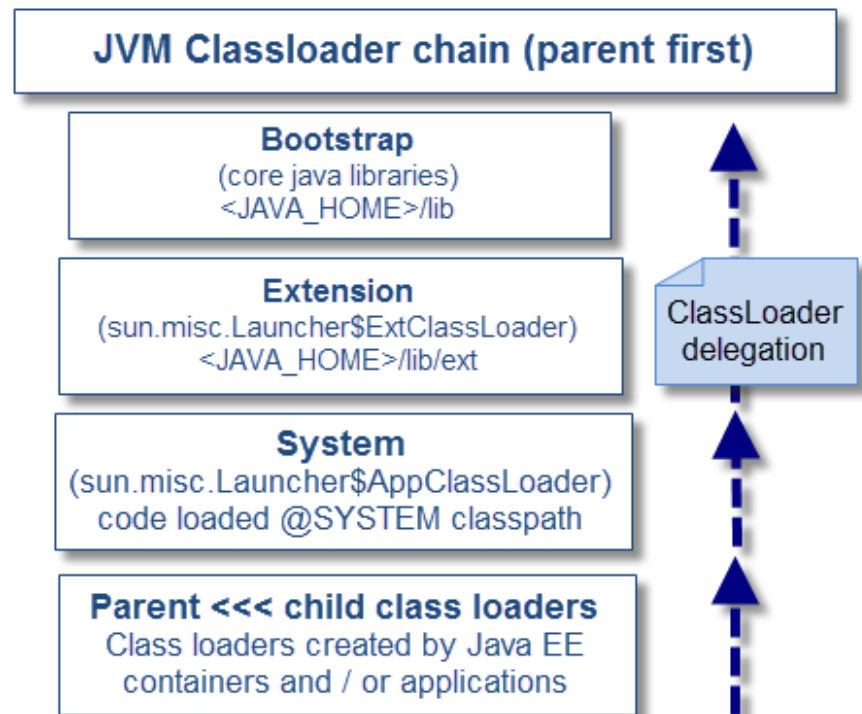
- ▶ `java Zoo`

Principe toujours facile pour les TP mais ce n'est pas la réalité !

L'enfer du classpath !

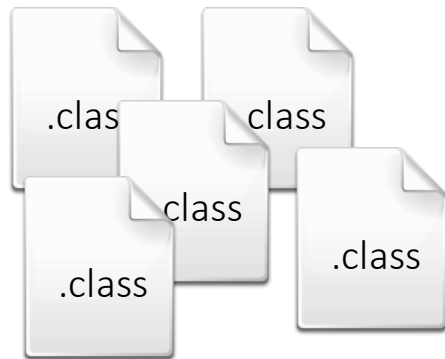
Exception in thread "main" java.lang.NoClassDefFoundError:

- ▶ Raison :
 - ▶ La classe n'est pas disponible dans le classpath
- ▶ Solution :
 - ▶ ~~Positionner la variable globale~~
 - OU ~~CLASSPATH~~
 - ▶ Utiliser l'option `-cp`



Pour livrer

► Que faut-il livrer ?



Ressources :

- Bibliothèques Java
- Fichiers image
- Fichiers de configuration (XML)
- ...



Idée : Fabriquer une archive contenant tout ce qui est nécessaire

Archive Java – 1/3

- ▶ Le format de fichier **Java Archive** (JAR) permet de packager de multiple fichiers en **un unique fichier d'archive**
 - ▶ Un fichier JAR contient généralement des **fichiers compilés** et des **ressources nécessaires** à une application
- ▶ Avantages des archives Java :
 - ▶ Compression : le format JAR permet de compresser efficacement les fichiers
 - ▶ Facilite l'extension de la plate-forme Java
 - ▶ Portabilité : le mécanisme de gestion des JAR est supporté sur toutes les plates-formes Java
 - ▶ Versionnement : un JAR contient de nombreuses informations comme le vendeur, le numéro de version...
 - ▶ Sécurité : les fichiers JAR peuvent être signés

Archive Java – 2/3

- Commandes utiles pour la manipulation d'une archive :

Opération	Commande
Création d'une archive non exécutable	<code>jar cvf jar-file input-file(s)</code> <code>jar cvf jar-file -C directory</code>
Création d'une archive exécutable	<code>jar cvfe jar-file MainClass input-file(s)</code> <code>jar cvfe jar-file MainClass -C directory</code>
Visualiser le contenu d'une archive	<code>jar tf jar-file</code>
Extraire le contenu d'une archive	<code>jar xf jar-file</code>
Exécuter une application packagée dans une archive non exécutable	<code>java -cp app.jar MainClass</code>
Exécuter une application packagée dans une archive exécutable	<code>java -jar app.jar</code>

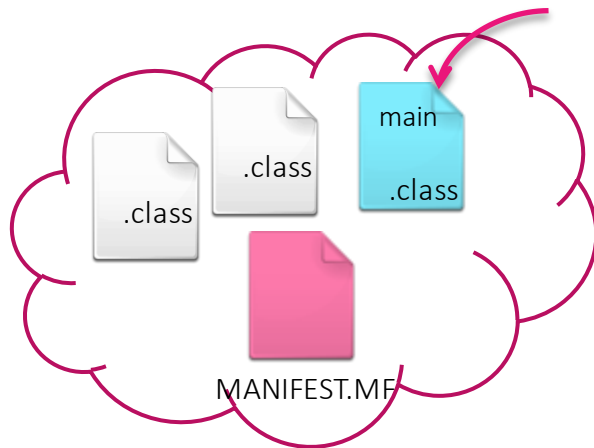
- Autres formats :

- WAR (Web Application Archive), EAR (Enterprise Application Archive), RAR (Resource Adapter Archive), AAR (Apache Axis Archive)...

Archive Java – 3/3

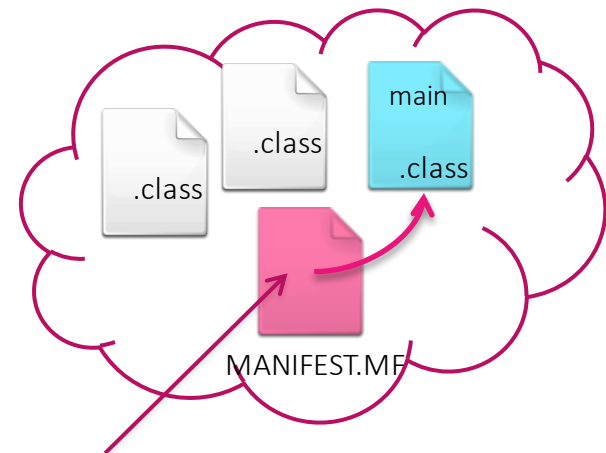
► Possibilité 1 :

- Archive non exécutable
- Nécessite à l'exécution de préciser la classe principale



► Possibilité 2 :

- Archive exécutable
- Ne nécessite pas à l'exécution de préciser la classe principale



Classe principale

Pour livrer

► Prérequis :

- Commande `jar` disponible dans le JDK (*Java Development Toolkit*)

► Packaging :

► Archive non exécutable :

- `jar cf zoo-v1.jar Zoo.class Animal.class Guepard.class`

► Archive exécutable :

- `jar cvfe zoo-v1.jar Zoo Zoo.class Animal.class Guepard.class`
- `jar cvfe zoo-v1.jar Zoo -C classes/ .`

Pour exécuter un JAR

- ▶ L'archive n'est pas exécutable :

```
Manifest-Version: 1.0  
Created-By: 1.8.0_171-b11 (Oracle Corporation)
```

MANIFEST.MF

- ▶ `java -cp zoo-v1.jar Zoo`

- ▶ L'archive est exécutable :

```
Manifest-Version: 1.0  
Created-By: 1.8.0_171-b11 (Oracle Corporation)  
Main-Class: Zoo
```

MANIFEST.MF

- ▶ `java -jar zoo-v1.jar`

Quelques définitions : JVM, JDK, JRE

► JVM :

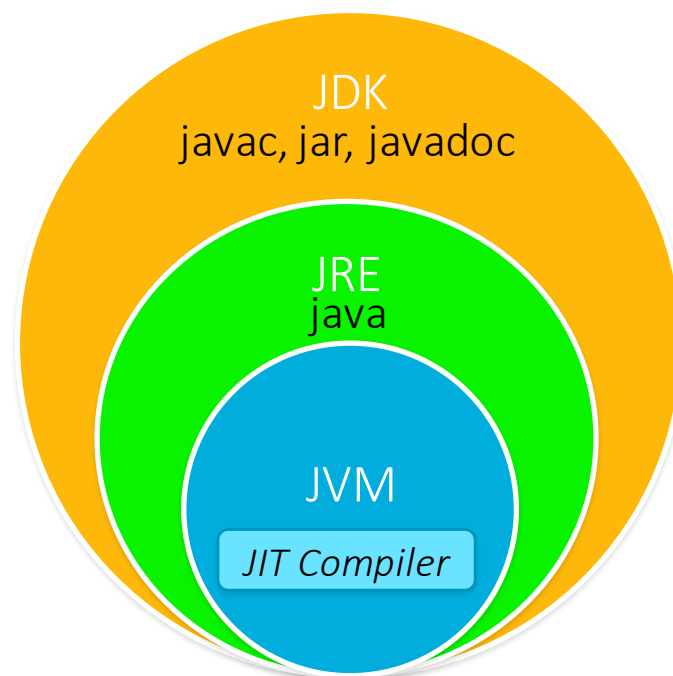
- La machine virtuelle Java (**Java Virtual Machine**) est un appareil informatique fictif qui exécute des programmes compilés sous la forme de bytecode Java.

► JRE :

- L'environnement de d'exécution (**Java Runtime Environment**) est un logiciel qui permet **l'exécution** des programmes écrits en langage Java (= JVM + bibliothèques)

► JDK :

- Le **Java Development Kit** est un ensemble de **bibliothèques** logicielles de base du langage Java ainsi que les **outils** avec lesquels le code Java peut être compilé, transformé en bytecode destiné à la JVM (= JRE + outils)



Que faut-il installer ?

► Attention :

- Nombreuses versions : 8, 9, 10, **11**, 12, 13, 14, 15, 16, 17, 18
- Formats différents : **SE** (Standard Edition), ME (Micro Edition), EE (Enterprise Edition)...

► 2 possibilités :

- JDK Oracle :
 - <https://www.oracle.com/fr/java/technologies/javase-jdk11-downloads.html>
- OpenJDK :
 - <https://jdk.java.net/archive/>

► **ATTENTION :**

- Par défaut, les machines personnelles ont toutes un JRE mais pas de JDK
- Il faut absolument installer un JDK !
- Risque de conflits entre le JRE par défaut et votre installation de JDK !

Configuration machine

► Prérequis :

- Installer un JDK

► Configuration des variables d'environnement :

- JAVA_HOME : pointe sur le répertoire du jdk
- PATH : pointe sur le répertoire /bin du jdk
 - Linux : `$PATH=$JAVA_HOME/bin:$PATH`
 - Windows : `%PATH%=%JAVA_HOME%\bin;%PATH%`

► Vérification de la configuration :

ET

- `java -version`
- `javac -version`



Doivent donner **EXACTEMENT**
le même numéro de version !

Automatisation des tâches

Apache Ant

Build Process en C

► Exemple : Makefile

```
CC = gcc
CFLAGS = -Wall -ansi -pedantic

all: main

main: acquisition.o stockage.o traitement.o main.o
    $(CC) $(CFLAGS) -o $@ $^%

.o: %.c %.h
    $(CC) $(CFLAGS) -c -o $@ $<

clean:
    rm *.o main data*
```

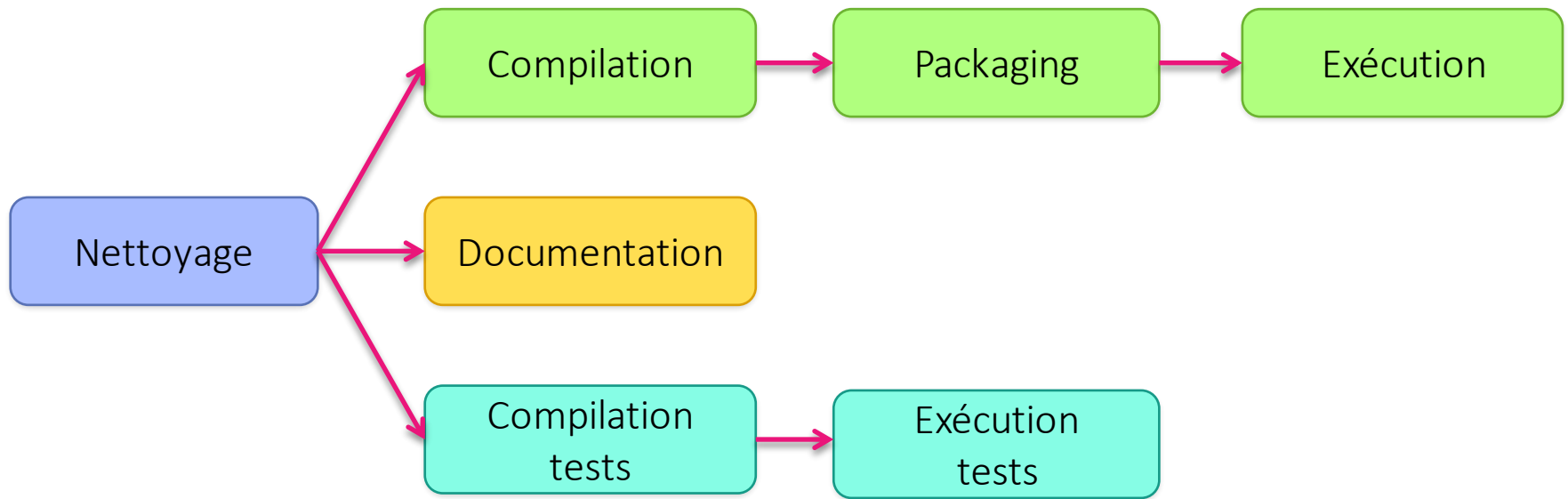


Build Process en Java – 1/2

- ▶ Quelles sont les tâches récurrentes à automatiser ?
 - ▶ Compilation
 - ▶ Packaging
 - ▶ Déploiement
 - ▶ Exécution
 - ▶ Tests
 - ▶ Documentation
 - ▶ ...

Build Process en Java – 2/2

- Existe-t-il un ordre pour l'exécution des tâches ?



Quels outils pour l'automatisation des tâches ?

Deux outils, deux philosophies

▶ Apache Ant

- ▶ <http://ant.apache.org/>
- ▶ Configuration à la charge du développeur pour faciliter la portabilité :
 - ▶ Ensemble de fonctionnalités qui ont le même comportement sur tous les systèmes
- ▶ Fichier build.xml



▶ Apache Maven

- ▶ <https://maven.apache.org/>
- ▶ Convention plutôt que configuration :
 - ▶ Maven impose une arborescence et un nommage des fichiers du projet
- ▶ Fichier pom.xml

maven

Apache Ant¹

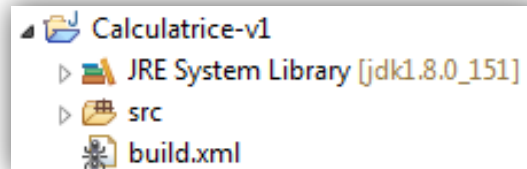


- ▶ Objectif :
 - ▶ **Automatiser les opérations répétitives** du développement logiciel telles que la compilation, la génération de documentation ou l'archivage au format Jar.
- ▶ Apache Ant est développé en Java
- ▶ Principalement utilisé pour des projets Java mais peut être utilisé pour tout autre type d'automatisation dans n'importe quel langage

¹ Acronyme de *Another Neat Tool* (« Un autre outil chouette »)

Fonctionnement de Apache Ant

- ▶ Description de l'ensemble des tâches qui doivent être effectuées de manière indépendante de la plate-forme d'exécution
 - ▶ Les tâches sont décrites en XML dans un fichier de configuration build.xml
- ▶ Intégration de Ant dans un projet Java
 - ▶ Ajout du fichier **build.xml** à la racine du projet



Structure d'un fichier build.xml

Nom du projet

Répertoire par défaut

Cible par défaut

Définition
des propriétés

Définition
des cibles
et des tâches

```
<project name="nomProjet" basedir="." default="cible1">

  <property name="var"      value="value"/>
  ...

  <target name="cible1">
    ...
  </target>
  <target name="cible2">
    ...
  </target>
  <target name="cible3" depends="cible2">
    ...
  </target>

</project>
```

Exemples de tâches simples

- ▶ Création d'un répertoire

```
<mkdir dir="rep"/>
```

- ▶ Suppression d'un fichier ou d'un répertoire

```
<delete file="file.jar"/>
```

```
<delete dir="rep"/>
```

- ▶ Copie d'un fichier ou d'un répertoire

```
<copy file="file.txt" tofile="myCopy.txt"/>
```

```
<copy file="file.txt" todir="../some/other/rep"/>
```

```
<copy file="file.txt" todir="../some/other/rep">  
  <fileset dir="src_dir" />  
</copy>
```


Exemples de tâches spécifiques à Java – 1/3

► Compilation : javac

Attribut	Rôle
srcdir	précise le répertoire racine de l'arborescence du répertoire contenant les sources
destdir	précise le répertoire où les résultats des compilations seront stockés
classpath	classpath pour l'exécution. Il est aussi possible d'utiliser un tag fils <classpath> pour le spécifier
classpathref	utilisation d'un classpath précédemment défini dans le fichier de build
target	précise la version de la plate-forme Java cible (1.1, 1.2, 1.3, 1.4, ...)
fork	lance la compilation dans une JVM dédiée au lieu de celle où s'exécute Ant. La valeur par défaut est false
source	version des sources Java : particulièrement utile pour Java 1.4 et 1.5 qui apportent des modifications à la grammaire du langage Java

► Exemple :

```
<javac srcdir="src_dir" destdir="dest_dir"/>
```

Exemples de tâches spécifiques à Java – 2/3

► Exécution : java

Attribut	Rôle
classname	nom pleinement qualifié de la classe à exécuter
jar	nom du fichier de l'application à exécuter
classpath	classpath pour l'exécution. Il est aussi possible d'utiliser un tag fils <classpath> pour le spécifier
classpathref	utilisation d'un classpath précédemment défini dans le fichier de build
fork	lancer l'exécution dans une JVM dédiée au lieu de celle où s'exécute Ant

► Exemple :

```
<java classname="test.Main">  
  <arg value="-h"/>  
  <classpath>  
    <pathelement location="dist/test.jar"/>  
  </classpath>  
</java>
```

Exemples de tâches spécifiques à Java – 3/3

► Packaging : jar

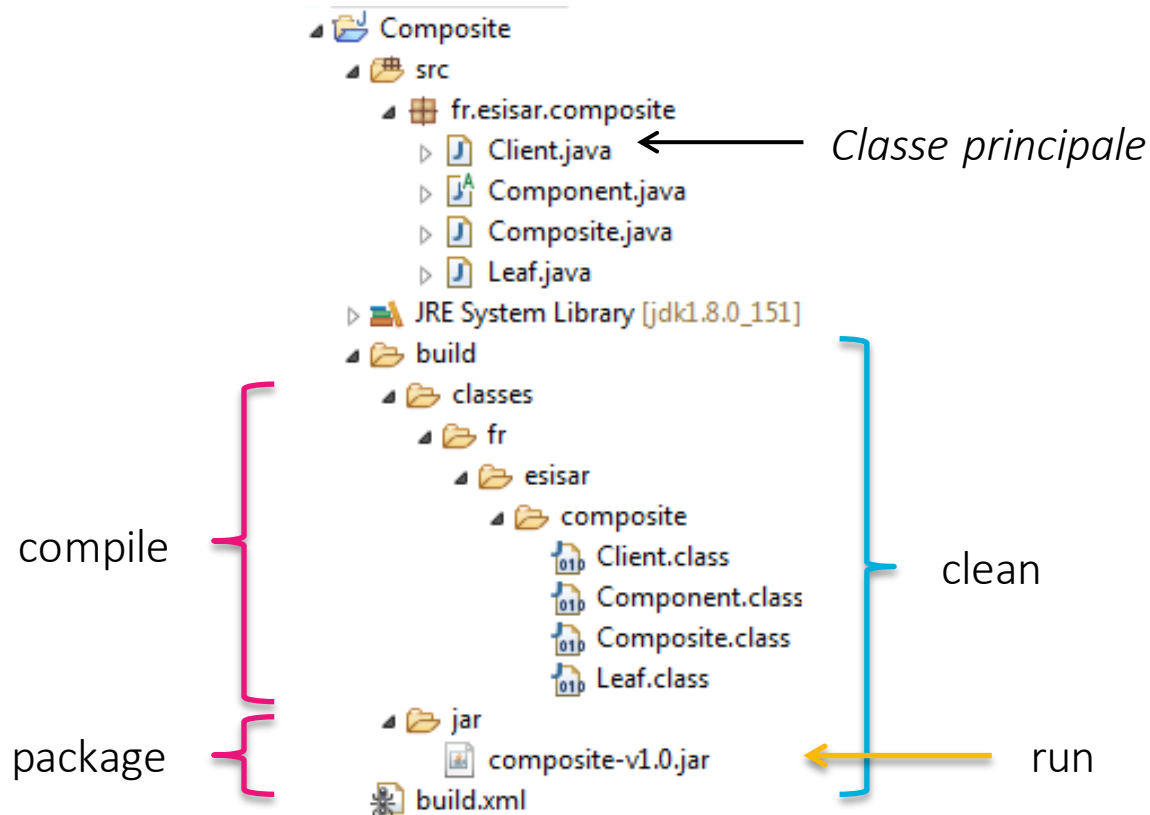
Attribut	Rôle
jarfile	nom du fichier .jar à créer
basedir	précise de répertoire qui contient les éléments à ajouter dans l'archive
manifest	précise le fichier manifest qui sera utilisé dans l'archive

► Exemples :

```
<jar destfile="monJar.jar"
    basedir="classes"
    includes="mypackage/test/**"
    excludes="**/Test.class" />
```

```
<jar destfile="monJar.jar" basedir="classes">
  <manifest>
    <attribute name="Main-Class" value="fr.esisar.MaClasse"/>
  </manifest>
</jar>
```

Exercice : écrire le fichier build.xml



Les propriétés

- ▶ Permet de définir la valeur d'une propriété qui pourra être ensuite utilisée plusieurs fois dans le projet

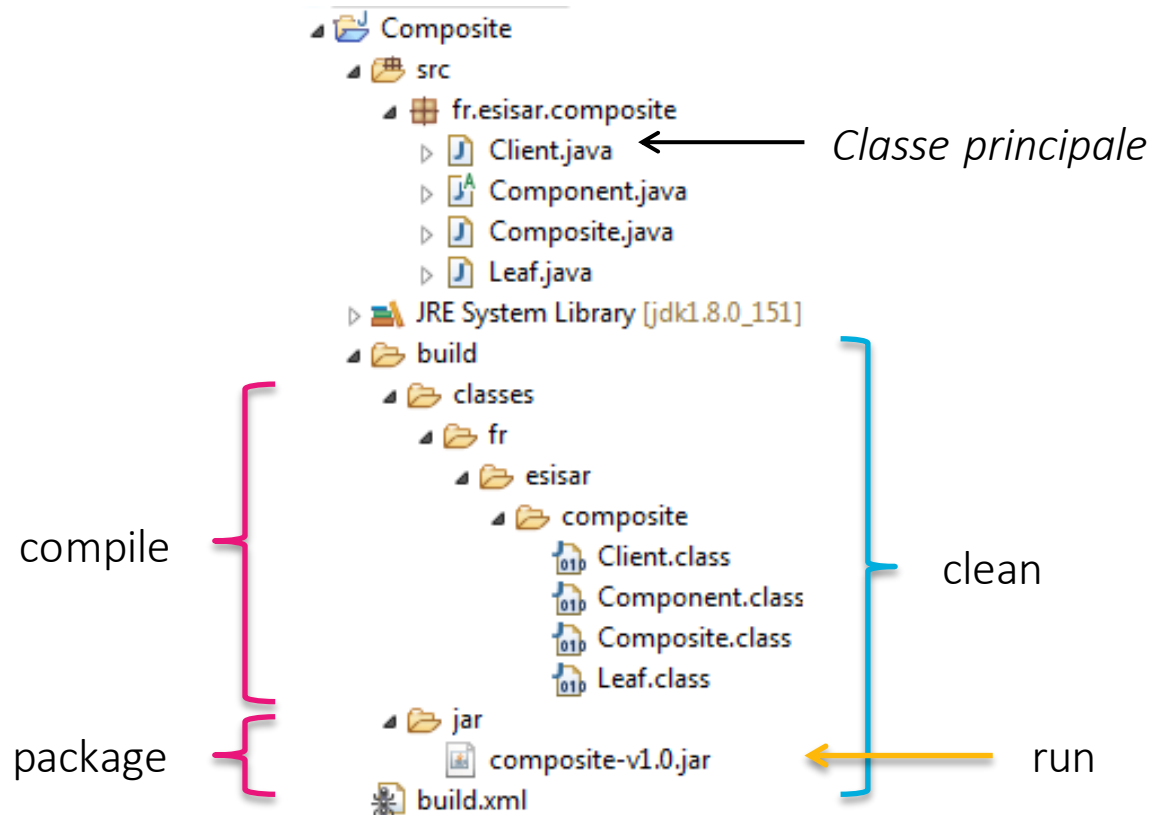
```
<property name="var" value="value"/>
```

- ▶ Utilisation d'une propriété : `${var}`

- ▶ Quelques propriétés pré-définies dans Ant :

basedir	chemin absolu du répertoire de travail (cette valeur est précisée dans l'attribut basedir du tag project)
ant.file	chemin absolu du fichier build en cours de traitement
ant.java.version	version de la JVM qui exécute ant
ant.project.name	nom du projet en cours d'utilisation

Exercice : build.xml avec des propriétés



- ▶ Objectif :
 - ▶ **Automatiser les opérations répétitives** du développement en général, plus particulièrement des projets Java EE, suivant les principes de l'intégration continue
- ▶ Principe :
 - ▶ Convention plutôt que configuration
 - ▶ Pour la structure du projet
 - ▶ Pour le cycle de vie (compilation, test, packaging, installation, déploiement)
- ▶ Basé sur un fichier pom.xml
- ▶ Utilise un dépôt d'archives (distant et local)

► Objectif :

- Automatisation des tâches pour construire des projets en Java, Scala, Groovy, Android, Kotlin, Swift, C++...

► Principe :

- Utilise les conventions à la manière de Maven
- Utilise la flexibilité de Ant pour décrire les tâches de construction