SCIENCES SORBONNE UNIVERSITÉ

# 2I002
# GESTION DE PROJETS
# PACKAGE, COMPILATION, IDE

Vincent Guigue

LIP6

MLIA Machine Learning & Deep Learning for Information Access

Bonne architecture = beaucoup de petites classes...
... chacune étant ciblée, lisible, ré-utilisable

$\Rightarrow$ Le répertoire de projet devient rapidement illisible !

**Solution = arborescence de répertoires**

- Sous-répertoires associés aux concepts de bas niveaux,
- Sous-sous-répertoires de test

# Exemple

Gestion d'une course de voiture autonomes

1. Réfléchir à un découpage de bas niveau:
   - **Circuit**
   - **Voiture**
   - Autonome $\Rightarrow$ gestion de l'**IA** / **stratégies**
2. Ajouter les outils (transverses)
   - Gestion de la **géométrie**
   - Gestion des fichiers (sauvegardes/chargements)
   - Interface graphique (IHM)

# Exemple

Gestion d'une course de voiture autonomes

1. Réfléchir à un découpage de bas niveau:
   - **Circuit**
   - **Voiture**
   - Autonome $\Rightarrow$ gestion de l'**IA** / **stratégies**
2. Ajouter les outils (transverses)
   - Gestion de la **géométrie**
   - Gestion des fichiers (sauvegardes/chargements)
   - Interface graphique (IHM)
3. Package de test:

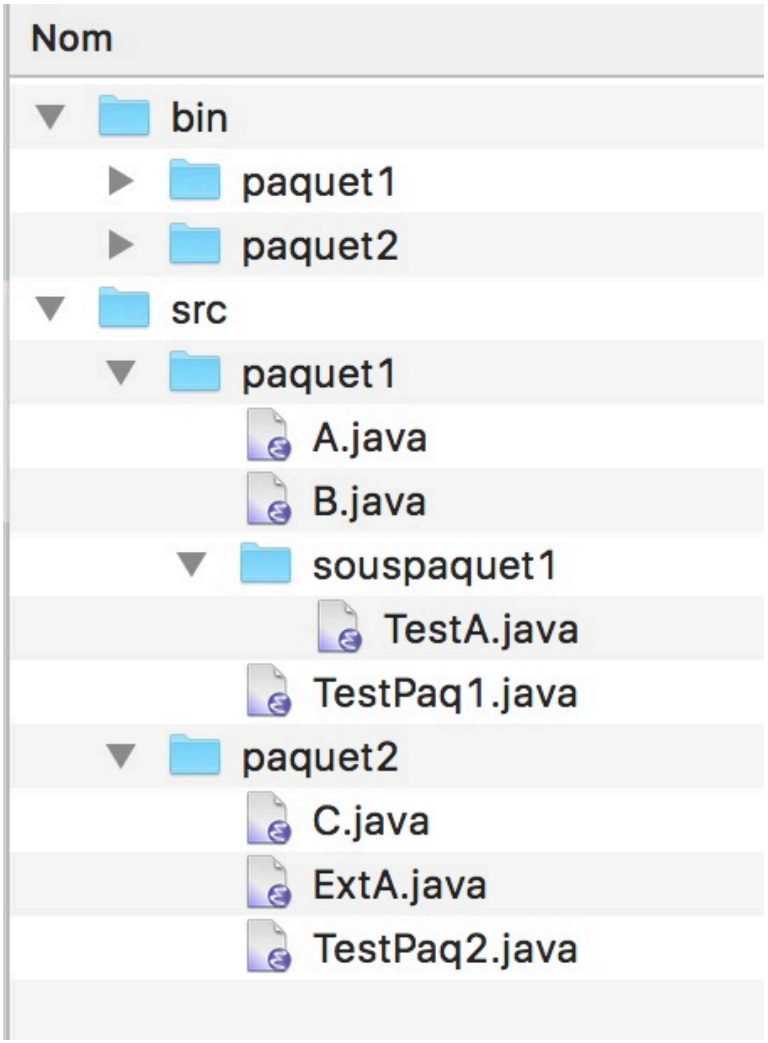
**Idée:**

valider le fonctionnement de chaque objet indépendamment du reste du projet (dans la mesure du possible).

$\Rightarrow$ **sous-répertoire de test** dans chaque package principal

# Déclarations obligatoires

Arborescence:




## 1 Déclaration de paquet

```java
// Fichier A.java
package paquet1;
public class A {
...
```

## 2 Déclaration d'import (pour les classes de paquets différents)

```java
package paquet2;
import paquet1.A;
public class ExtA extends A{
    public ExtA() {
        super();
    }
```

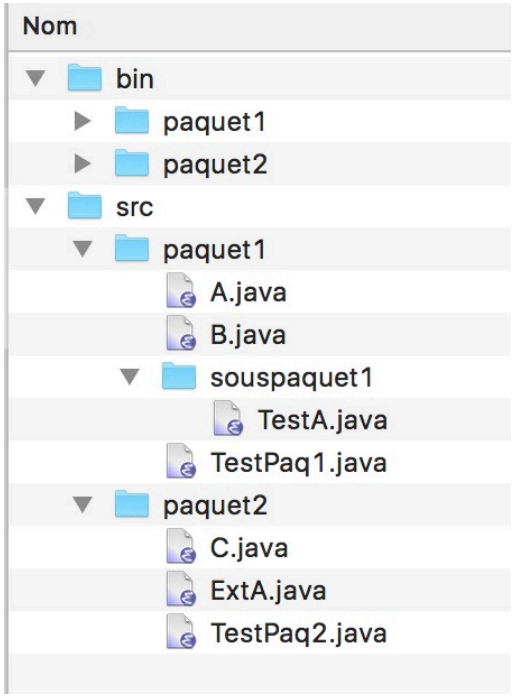
## 3 Sous-package

```java
package paquet1.souspaquet1;
public class TestA {
    public static void main(String[] args) {
        // tests specifiques a A
```

## 4 Classe JDK

```java
import java.util.ArrayList;
```

# Compilation / exécution du code




- **Compilation** (position = racine)
  - Spécification d'un répertoire cible : `-d`
  - Spécification du répertoire de gestion des sources : `-cp`

```
» javac -cp src -d bin src/paquet1/TestPaq1.java
```

⇒Compile l'exécutable + toutes les dépendances

- **Exécution**
  - Instruction pour se positionner dans le répertoire d'exécution: `-cp`
  - Chemin avec des `.` (pas des `/`)

```
» java -cp bin paquet1.TestPaq1
```

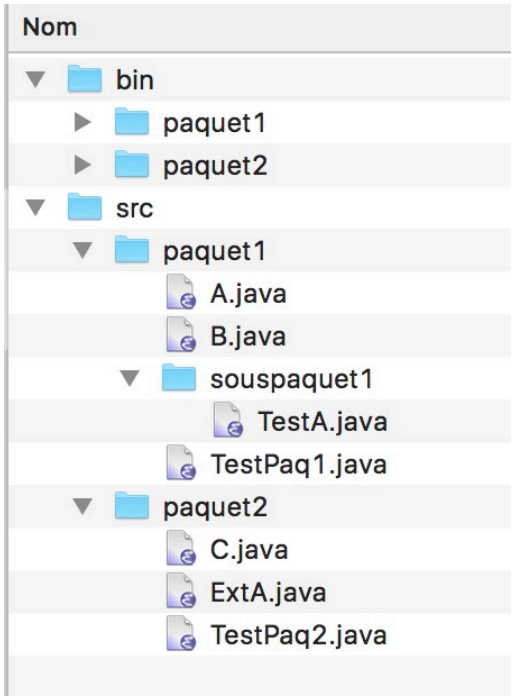
ou

```
» cd bin
» java paquet1.TestPaq1
```

SCIENCES SORBONNE UNIVERSITÉ

## introduction des packages = subtilités sur la visibilité

```
1  package paquet1;
2  public class A {
3    public int i;      // public
4    protected int j;   // protected
5    private int k;     // private
6    int n;             // package (nouveau)
7
8    public A(){
9        i=1; j=2; k=3; n=4;
10   }
11 }
```




Visibilités des attributs de `A` depuis :

| | | i | j | k | n |
|---|---|---|---|---|---|
| Même répertoire | `B, TestPaq1` | √ | √ | × | √ |
| Classe fille | `ExtA` | √ | √ | × | × |
| Autres cas | `C,TestPaq2,TestA` | √ | × | × | × |