ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR

Membre de HONORIS UNITED UNIVERSITIES


# LES PACKAGES

## 3IIR - POO2

# PACKAGES : UTILITÉ

- **Organiser les classes de manière hiérarchique.**
  - Généralement, les classes qui vont ensemble sont regroupées dans le même **package**.
- **Eviter les conflits de noms**
  - Les packages permettent de garantir que le nom que vous choisissez pour une classe sera unique et n'entrera pas en conflit avec les noms de classes choisis par d'autres programmeurs.
- **Faciliter le chargement des classes**
  - La JVM peut charger les classes plus efficacement grâce à une structure de nommage bien définie.

# PACKAGES : AVANTAGES

- **Les packages permettent de réutiliser et de partager facilement du code entre différents projets.**
  - Vous pouvez créer une bibliothèque de classes dans un package que vous mettez à disposition pour d'autres développeurs ou applications.
- **Les packages jouent un rôle dans la gestion de la visibilité des classes, méthodes et attributs qui peuvent avoir des niveaux d'accès différents.**
  - Vous pouvez définir des classes, à l'intérieur d'un package, qui ne sont pas accessibles par du code extérieur à ce package.
  - Vous pouvez également définir des membres de classe qui sont visibles uniquement par les autres classes du même package.

# TYPES DE PACKAGES

- **Packages standards :** comme **java.lang**, **java.util**, etc.
  - Sont fournis par le JRE et contiennent des classes et des interfaces que vous pouvez utiliser dans vos applications
- **Packages personnalisés :** que vous créez pour regrouper et organiser les classes de votre application.

# DÉFINIR UN PACKAGE

- Un **package** est défini avec le mot-clé **package**.
  - `Syntaxe : package nom_package;`
    - **Cette commande doit être la première instruction dans un fichier source Java.**
    - **Toutes les classes définies dans ce fichier appartiendront au package spécifié.**
    - **Le nom du package et de la classe doivent former une combinaison unique pour éviter les conflits de noms.**

# NOMMER UN PACKAGE

- Utilisez des noms en minuscules et suivez une **structure hiérarchique unique et facile à comprendre**
  - **En général, l*e nom du package est l'inverse du nom de domaine de votre entreprise, plus le système de dénomination adopté au sein de votre entreprise.***
- **`Exemple :  ma.emsi.ventes;`**
  - Le domaine est **emsi.ma**
  - La structure interne de dénomination de packages est liée aux **noms des départements**.
  - Le nom du package pour un projet développé par le département des **ventes** pourrait être :

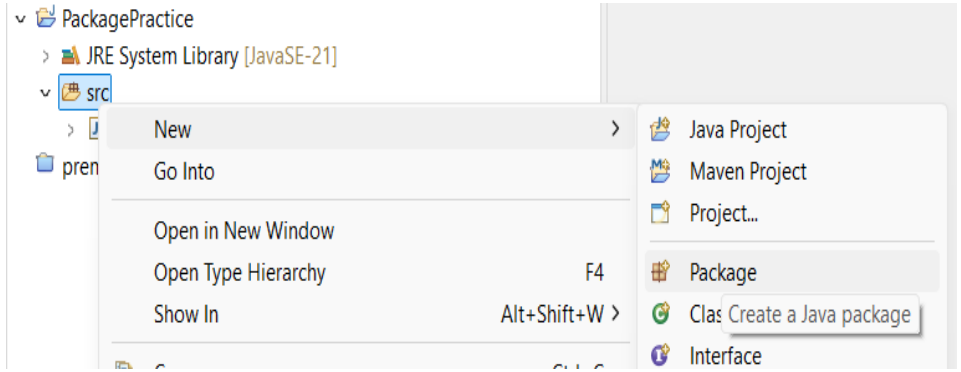
```
ma.emsi.ventes;
```

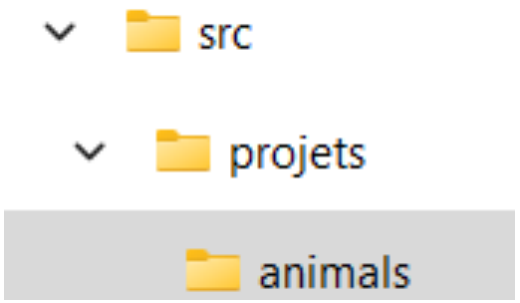
# STRUCTURE D'UN PACKAGE

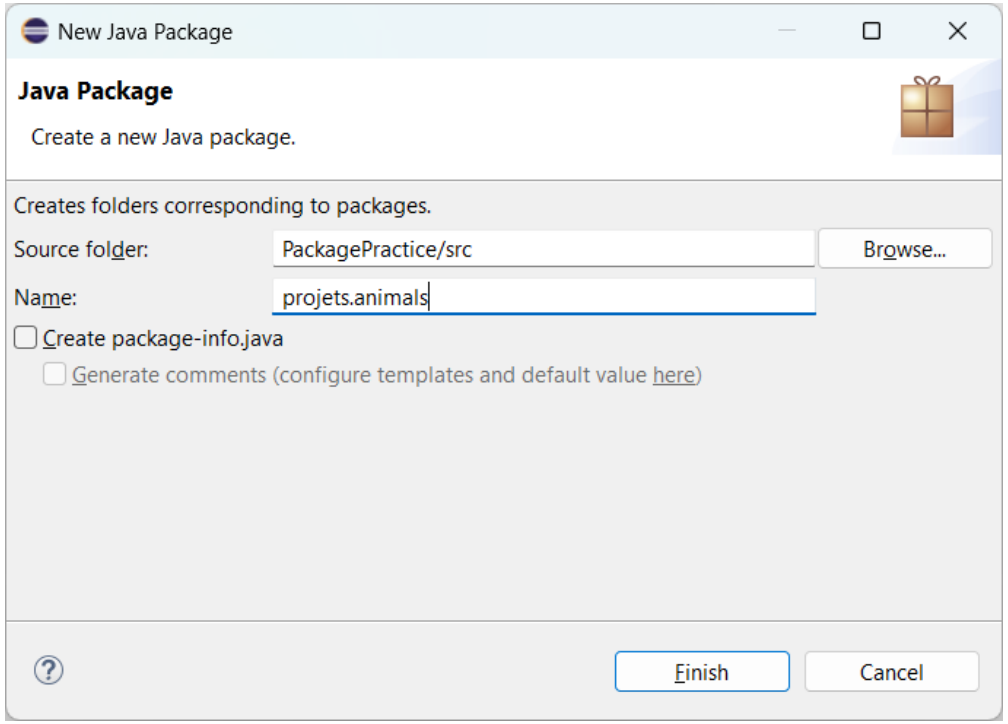
- **Java utilise la structure du système de fichiers pour organiser les packages et les classes de manière hiérarchique.**
  - Le package **`ma.emsi.ventes`** sera représenté par un répertoire **ma/emsi/ventes** dans votre système de fichiers, et les fichiers sources, des classes de ce package, se trouveront dans ce répertoire.
  - Les fichiers **.class** pour toutes les classes que vous déclarez faire partie de **`ma.emsi.ventes` doivent être stockés** dans un répertoire dont le chemin se termine par **ma/emsi/ventes**

# STRUCTURE D'UN PACKAGE

- La création d'un package en **Eclipse** et **IntelliJ** se traduit par la création de répertoires dans le dossier **src ,** suivant la structure hiérarchique du package.

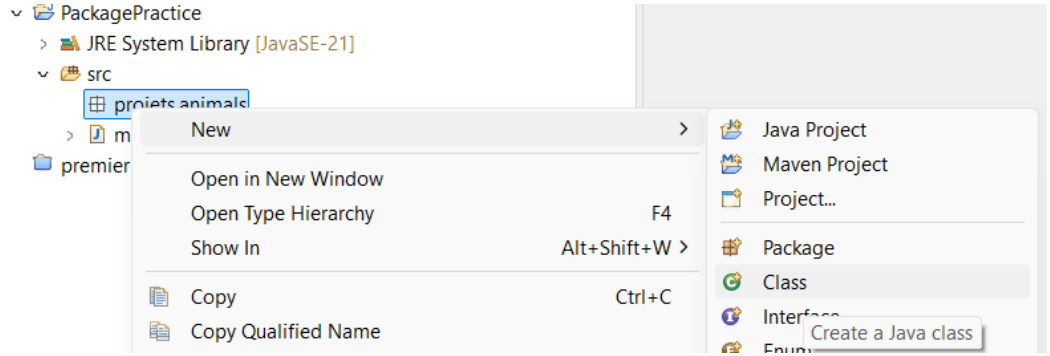


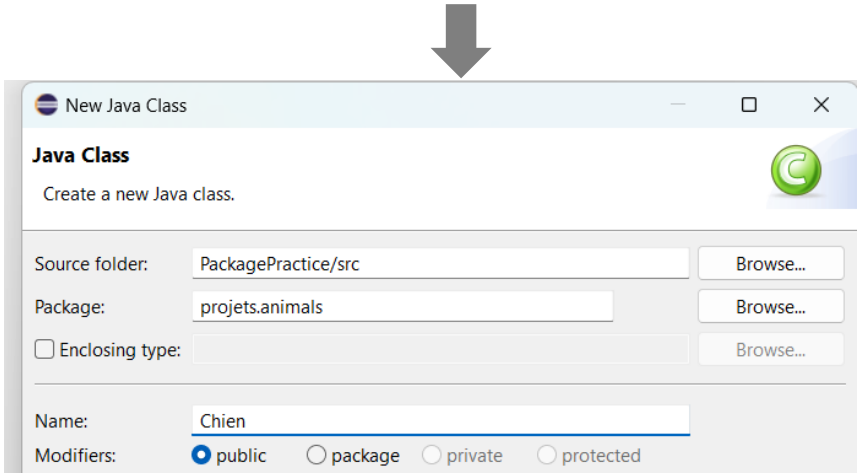



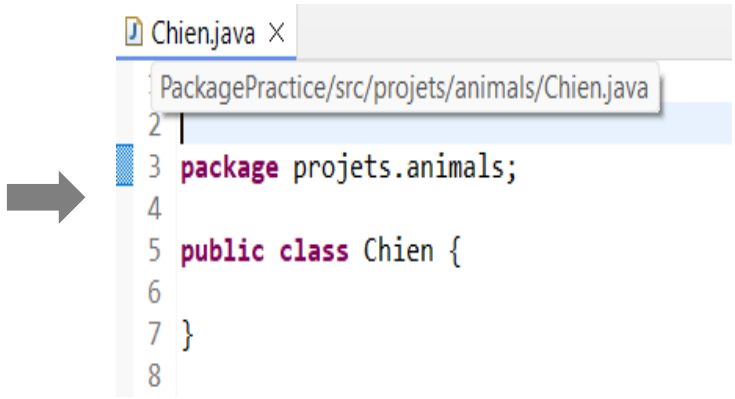
# STRUCTURE D'UN PACKAGE

- En **Eclipse** et **IntelliJ**, un fichier source Java (.java) est placé dans un répertoire **src** en respectant la hiérarchie du package.



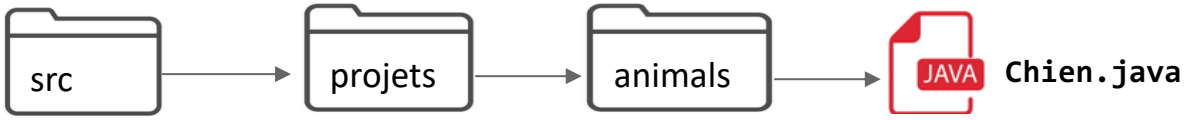

```
src
└── projets
    └── animals
        ├── Chien.java
        ├── ...
```



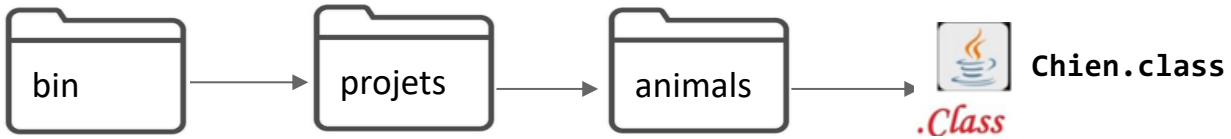
# STRUCTURE D'UN PACKAGE

- Si la première ligne du fichier source d'une classe **Chien** est : ***package projets.animals;***
  - La classe ***Chien*** se trouve dans le package ***projets.animals***
  - Le nom du fichier source est ***Chien.java*** et il se trouve dans le dossier ***projets/animals***

```
package  projets.animals;
public class   Chien
{
      String nom;
}
```




  - Le **nom complet** de la classe ***Chien*** est ***projets.animals.Chien***
  - La fin du chemin d 'accès du fichier compilé, **Chien.class,** doit être ***projets/animals***



# UTILISER UNE CLASSE D'UN PACKAGE

- **Par défaut, une classe d'un package, n'est accessible qu'aux classes du même package.**
- Pour utiliser une classe dans un autre package :
  - La classe doit être déclarée ***public.***
  - On doit utiliser son nom complet sous la forme ***nompackage.nomclasse***
- ***Exemple :***

```
package projets.personnes;

public class Owner {

String nom;

}
```

**Owner.java**

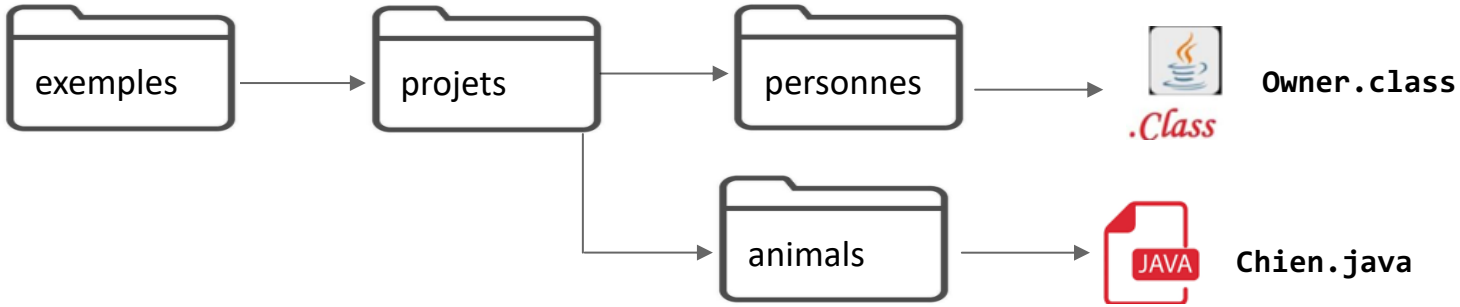
```
package projets.animals;
public class Chien
{
     String nom;

     projets.personnes.Owner O = new projets.personnes.Owner();

}
```

**Chien.java**

# IMPORTER UNE CLASSE

- Pour éviter d'écrire le nom complet de la classe, utilisez l'instruction ***import*** .
  - **Pour importer une classe spécifique.**
    - **Exemple** : `import java.util.Scanner;`
  - **Pour importer toutes les classes d'un package en utilisant ***
    - **Exemple :** `import java.util.*;` **=> importe uniquement les classes du package java.util pas celles de ses sous-packages.**
- L'instruction ***import*** n'importe rien au sens propre. Ce qu'elle fait réellement, c'est ***permettre d'utiliser le nom court d'une classe au lieu de son nom complet.***

# IMPORTER UNE CLASSE




- L'exemple suivant montre deux manières d'éviter d'utiliser le nom complet de la classe ***Owner*** grâce à ***import.***
  - **L'instruction *import* s'écrit après la définition du package et avant la définition de la classe.**

```
package  projets.animals;
import projets.personnes.Owner;
class   Chien
{
   String nom;
   Owner O = new Owner();
}
```

Chien.java

```
package  projets.animals;
import projets.personnes.*;
class   Chien
{
   String nom;
   Owner O = new Owner();
}
```

Chien.java

# TROUVER UN PACKAGE : CLASSPATH

- Par défaut, le système d'exécution Java utilise le répertoire de travail actuel comme point de départ pour chercher un package.
  - Si votre package se trouve dans un sous-répertoire du répertoire actuel, il sera trouvé.
  - Sinon, utilisez l'option **–classpath** avec **java** ou **javac** pour spécifier le chemin où Java cherchera les fichiers .class nécessaires à la compilation.

# TROUVER UN PACKAGE : CLASSPATH

```
package projets;
import projets.animals.Chien;
class Test {

    Chien fox = new Chien();

}
```

```
src
└── projets/
    ├── Test.java
    └── animals/
        └── Chien.java

bin
└── projets/
    ├── Test.class
    └── animals/
        └── Chien.class
```

- Pour compiler Test.java , à partir du répertoire "**src/projets**" , où se trouve le fichier "Test.java", on lance la commande :

```
> javac  -cp ../../bin Test.java
```

- Le répertoire de travail actuel est **src/projets.** On veut utiliser le fichier **Chien.class** qui se trouve dans **bin/projets/animals**
- À l'aide de **–cp ,** on précise le chemin nécessaire pour accéder au répertoire **bin.** Et automatiquement guidé par le **import** , Java cherchera le fichier **Chien.class** dans **projets/animals.**

# TROUVER UN PACKAGE : CLASSPATH

```
src
└── projets/
    ├── Test.java
    └── animals/
        └── Chien.java

bin
└── projets/
    ├── Test.class
    └── animals/
        └── Chien.class
```

- Pour compiler Test.java , à partir du répertoire "**src**" , on lance la commande :

```
> javac -cp ../bin projets.Test.java
```

- Le répertoire de travail actuel est **src.** On veut utiliser le fichier **Chien.class** qui se trouve dans **bin/projets/animals**
- À l'aide de **–cp ,** on précise le chemin nécessaire pour accéder au répertoire **bin.** Et automatiquement guidé par le **import** , Java cherchera le fichier **Chien.class** dans **projets/animals**