Packages et modules

dernière modification 14/01/2023

Philippe Genoud
Philippe.Genoud@imag.fr

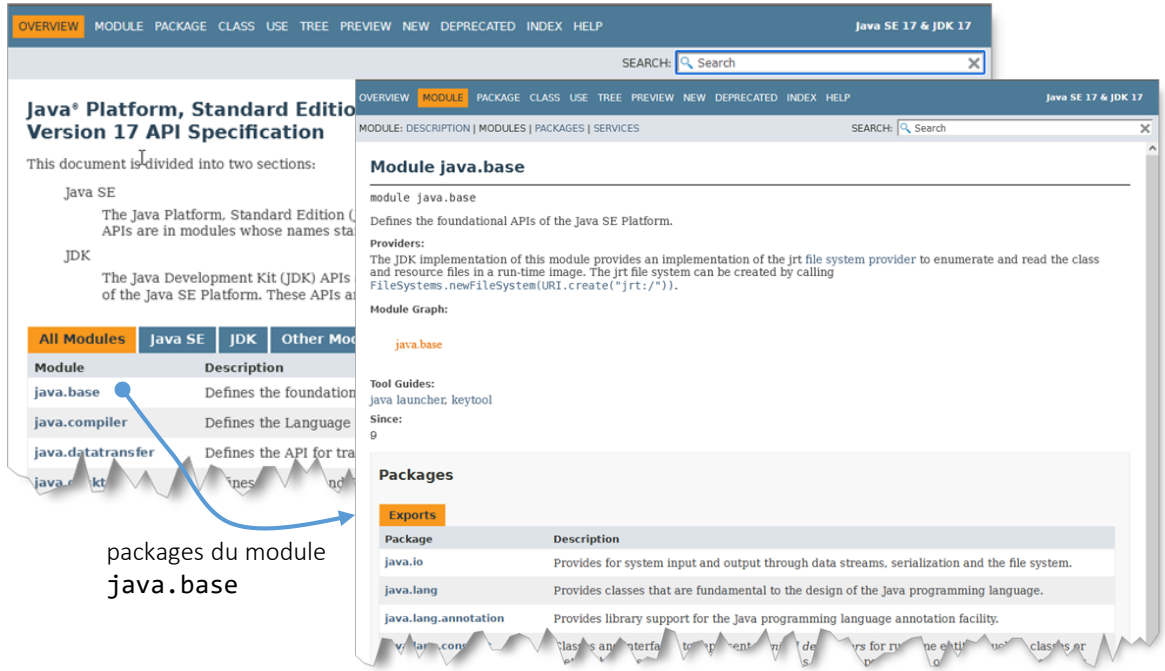
Notion de package

- un package est un groupe de classes associées à une fonctionnalité et/ou qui coopèrent

Java 8 https://docs.oracle.com/javase/8/docs/api/

Java 9+ : https://docs.oracle.com/en/java/javase/17/docs/api/

un niveau de structuration supplémentaire les modules

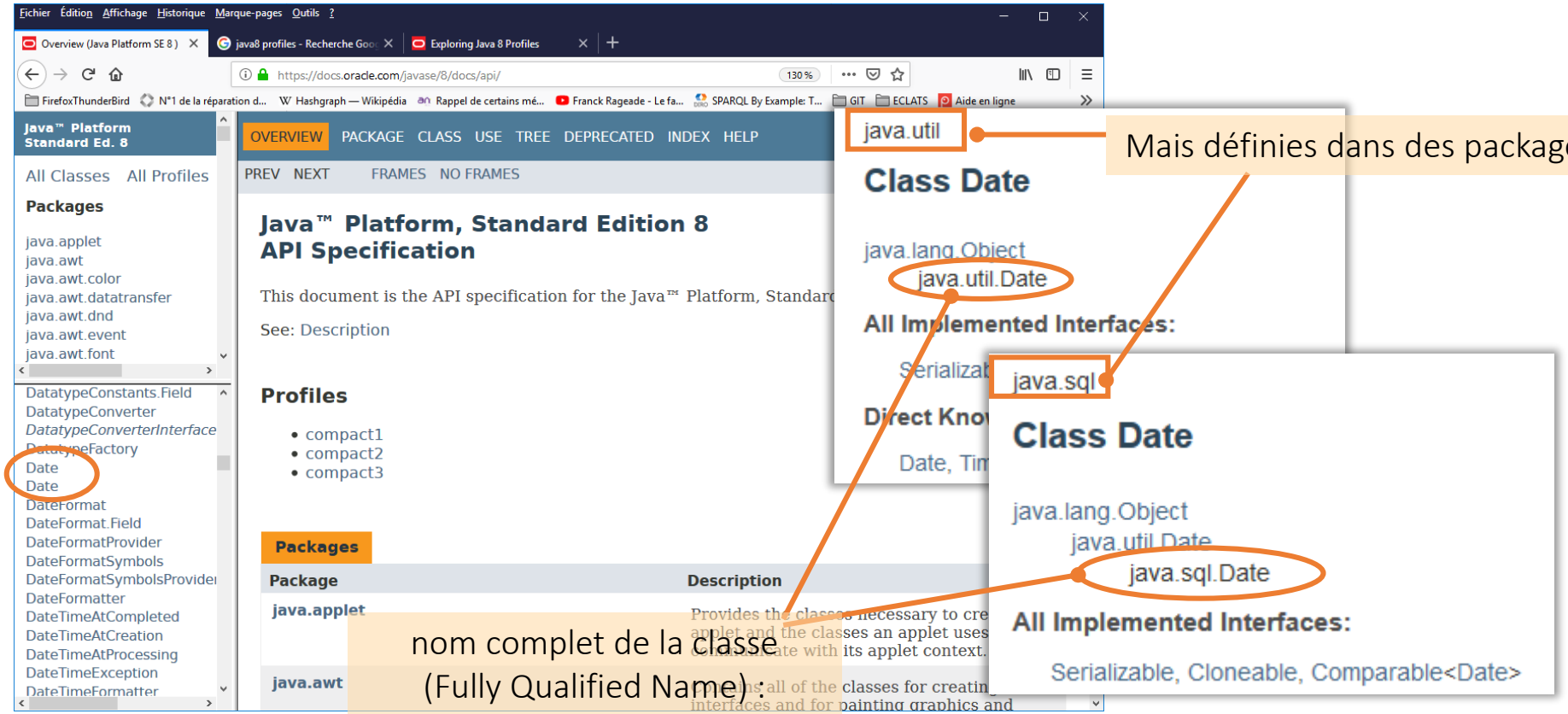


packages du module `java.base`

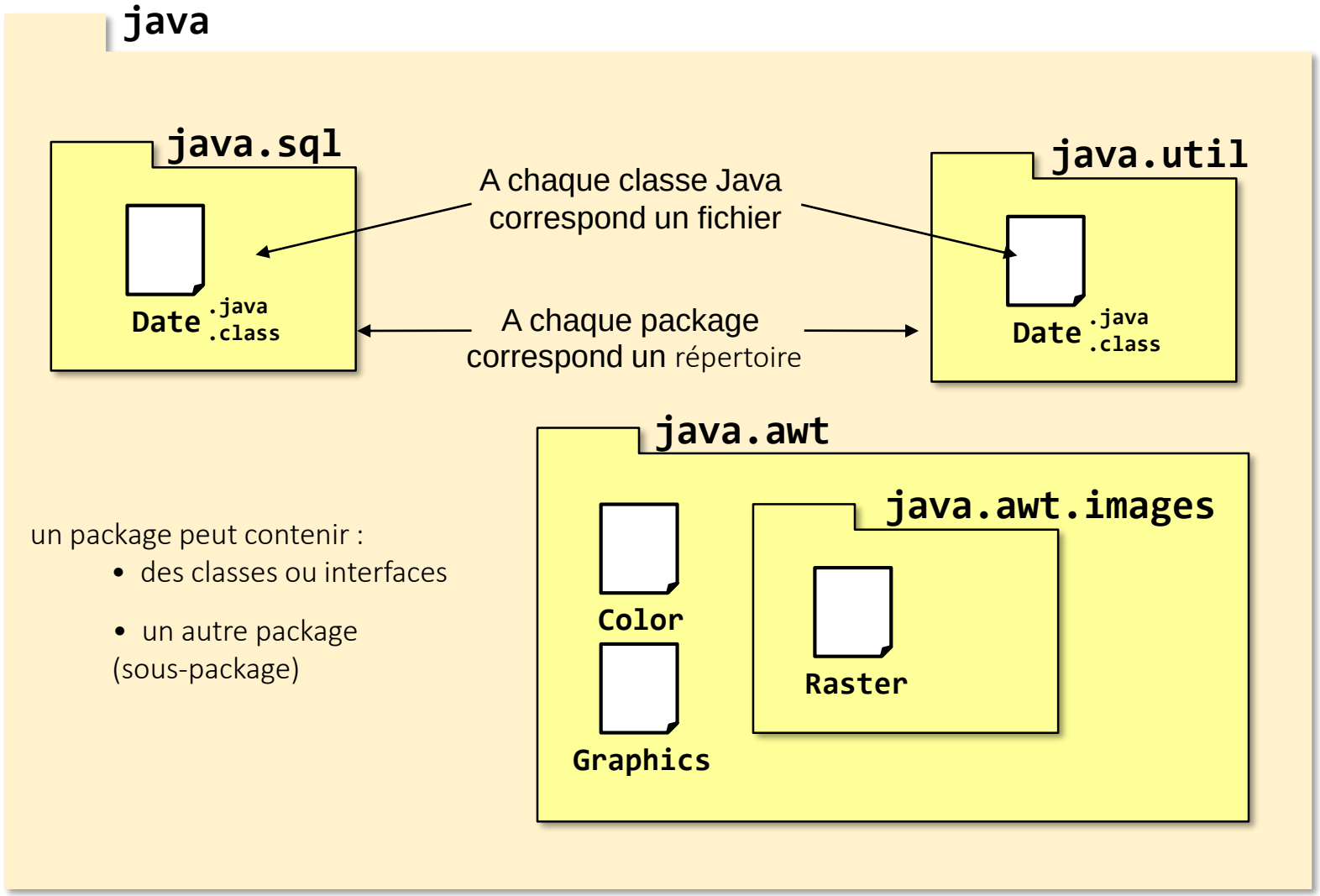
- exemples de packages
  - **`java.lang`** : rassemble les classes de base JAVA, **`Object`**, **`String`**, **`System`**...
  - **`java.util`** : classes utilitaires (`Scanner`, `Date` ...) + classes pour les collections
  - **`java.awt`** : classes pour interfaces utilisateurs (Abstract Window Toolkit)
  - **`java.awt.images`** : classes pour manipulation d'images bitmap
  - ....

Package : espace de nommage

- Le regroupement des classes dans des packages permet d'organiser les librairies de classes Java et d'éviter d'éventuels conflits de noms.
  - Exemple : dans les librairies standards du JDK deux classes **`Date`**



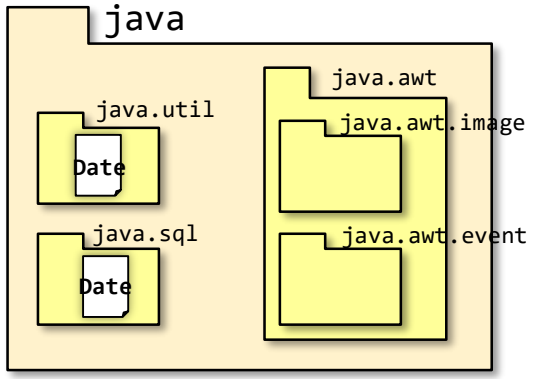
Packages et système de fichiers

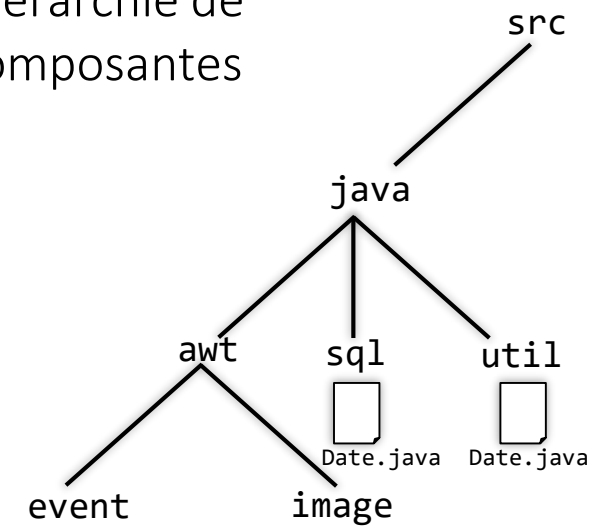


La hiérarchie des packages correspond à une hiérarchie de répertoires

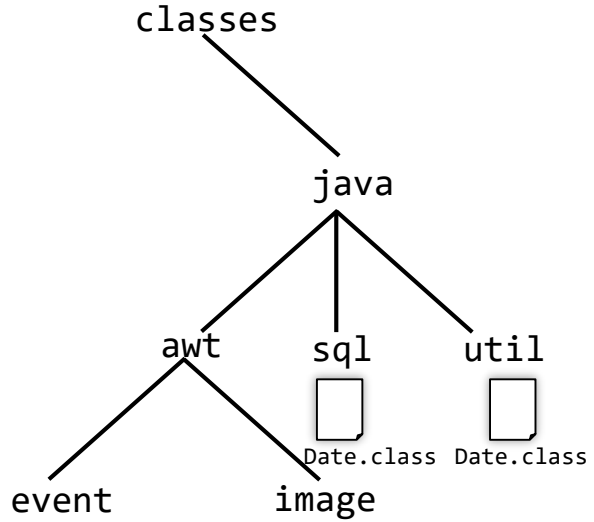
Packages et système de fichiers

- A une hiérarchie de packages correspond une hiérarchie de répertoires dont les noms coïncident avec les composantes des noms de package



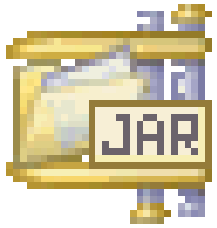


Au niveau des sources (`.java`)



Au niveau du bytecode (`.class`)

Dans les fichiers jar



`rt.jar`

```
C:\Program Files\Java\jdk1.8.0_231\jre\lib
λ jar tvf rt.jar | more
     0 Sat Oct 05 03:32:58 CEST 2019 META-INF/
  2341 Sat Oct 05 03:32:58 CEST 2019 META-INF/MANIFEST.MF
   754 Sat Oct 05 03:19:50 CEST 2019 com/oracle/net/Sdp$1.class
   350 Sat Oct 05 03:19:50 CEST 2019 com/oracle/net/Sdp$SdpSocket.class
  3054 Sat Oct 05 03:19:50 CEST 2019 com/oracle/net/Sdp.class
  2181 Sat Oct 05 03:20:42 CEST 2019 com/oracle/nio/BufferSecrets.class
   530 Sat Oct 05 03:20:42 CEST 2019 com/oracle/nio/BufferSecretsPermission.class
   379 Sat Oct 05 03:20:42 CEST 2019 com/oracle/util/Checksums.class
  1840 Sat Oct 05 03:15:20 CEST 2019 com/oracle/webservices/internal/api/EnvelopeStyle$Style.class
   744 Sat Oct 05 03:15:20 CEST 2019 com/oracle/webservices/internal/api/EnvelopeStyle.class
   975 Sat Oct 05 03:15:20 CEST 2019 com/oracle/webservices/internal/api/EnvelopeStyleFeature.class
  1465 Sat Oct 05 03:15:18 CEST 2019 com/oracle/webservices/internal/api/databinding/Databinding$Builder.class
   963 Sat Oct 05 03:15:18 CEST 2019 com/oracle/webservices/internal/api/databinding/Databinding.class
  1962 Sat Oct 05 03:15:18 CEST 2019 com/oracle/webservices/internal/api/databinding/DatabindingFactory.class
```

Instruction package

conventions :
- minuscules pour les noms de packages
- les préfixer par nom de domaine inversé ex : `org.junit.jupiter` pour JUnit (junit.org) `jena.apache.org` pour Apache Jena (jena.apache.org)

- Le package d'appartenance d'une classe est défini par l'instruction `package nom.du.package;`

Instruction **optionnelle**
Si elle est présente elle doit être la **première** instruction du fichier,
elle définit le package auquel est rattaché le code contenu dans le fichier.

```
package courspoo;
public class UneClasse {
}
```

```
package courspoo.banque;
public class Compte {
}
```

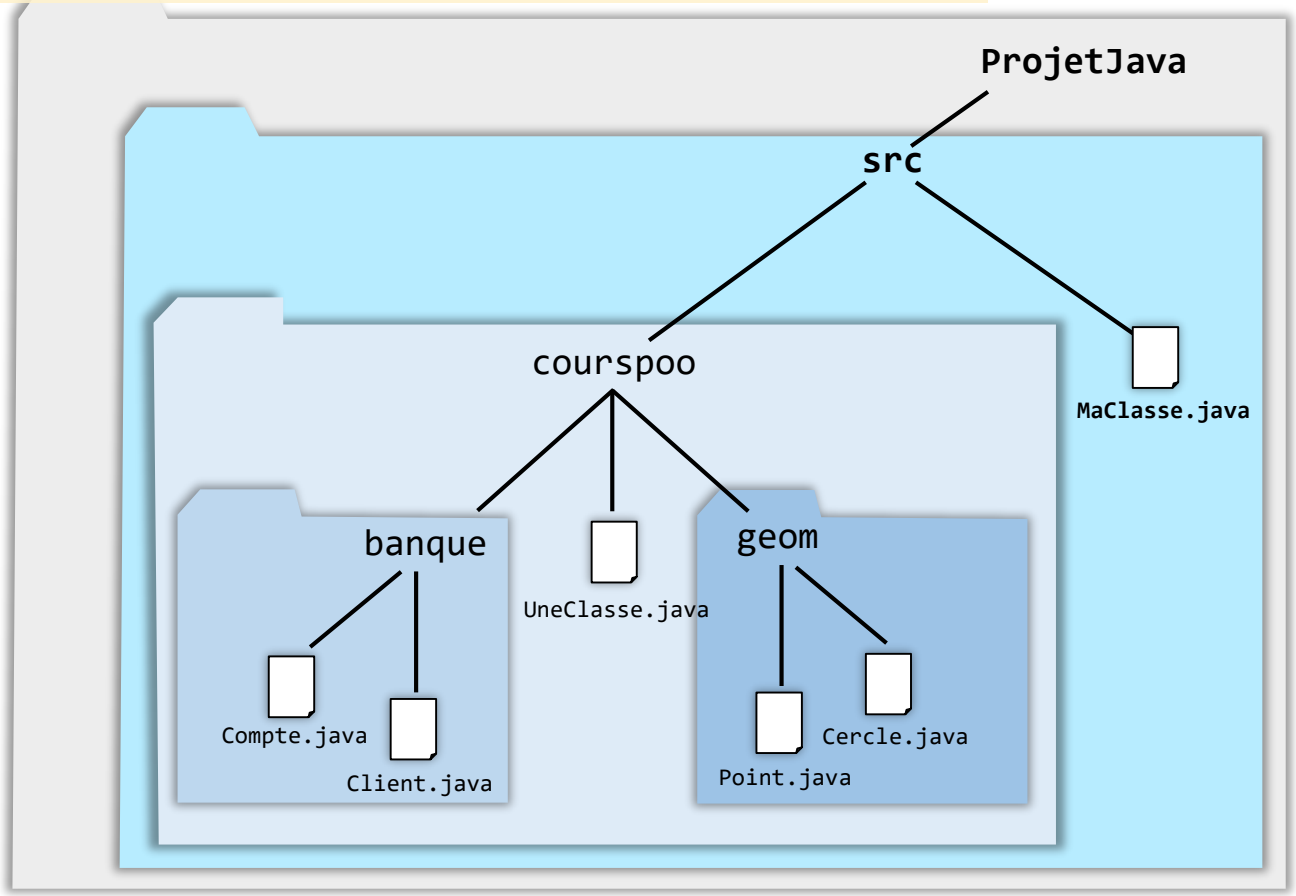
```
package courspoo.banque;
public class Client {
}
```

```
package courspoo.geom;
public class Point {
}
```

```
package courspoo.geom;
public class Cercle {
}
```

```
public class MaClasse {
    ...
}
```

Si pas d'instruction package : le code défini dans le fichier est associé à un package par défaut, sans nom et correspondant au répertoire racine des sources



Accès aux éléments d'un package

- Exemple : dans une classe définie dans le package `courspoo.banque` on souhaite utiliser un objet de type `Connection`, classe du package standard `java.sql` qui permet de faire une connexion réseau à une base de donnée relationnelle.

```
package courspoo.banque;



public class Compte {

    private double solde;
    private String titulaire

    private Connection c; 



    ...
 }
```

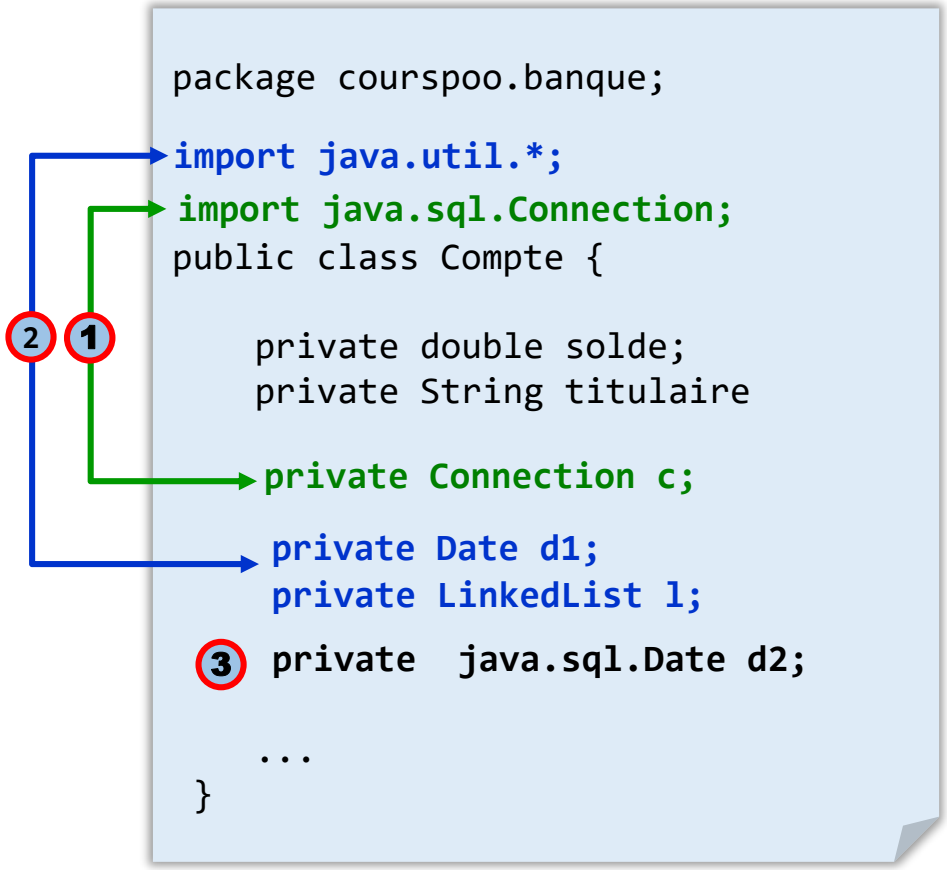
Compilation de la classe

```
javac courspoo.banque.Compte
```

```
courspoo/banque/Compte.java [32:1] cannot resolve symbol
symbol  : class Connection
location: class courspoo.compte.Compte
  private Connection s;
          ^
1 error
Errors compiling Compte.
```

Accès aux éléments d'un package

- Pour pouvoir utiliser une classe issue d'un autre package il faut signifier son origine au compilateur (instruction **`import`**)



1. en important la classe

```
import nomDuPackage.NomDeClasse;
```

2. en important tout le package où est définie la classe

```
import nomDuPackage.*;
```

3. en indiquant le nom complet de la classe (Fully Qualified Name).

> **`import`** n'insère pas du code dans le code de la classe (comme le fait **`#include`** du C). Il s'agit simplement d'une indication pour le compilateur. En interne seuls les FQN (Fully Qualified Names) des classes sont utilisés.

Pas besoin d'import pour les classes du package **`java.lang`**

De manière générale on préférera la solution 1 à la solution 2

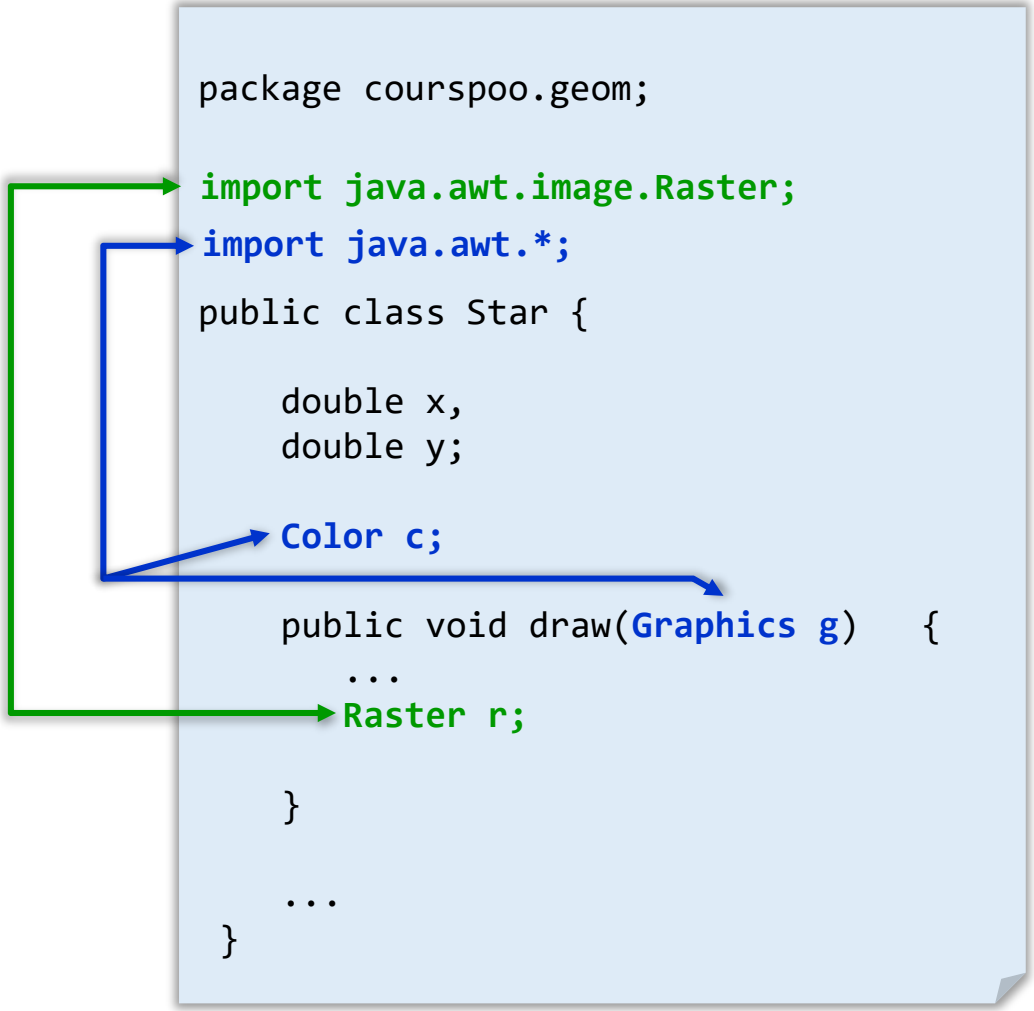
```
import java.util.Date;
import java.util.LinkedList;
```

plutôt que

```
import java.util.*;
```

Accès aux éléments d'un package

- L'instruction `import nomPackage.*` ne concerne que les classes du package indiqué. Elle ne s'applique pas aux classes des sous packages.



```
package courspoo.geom;

import java.awt.image.Raster;
import java.awt.*;

public class Star {

    double x,
    double y;

    Color c;

    public void draw(Graphics g)   {
       ...
    Raster r;

    }

    ...
 }
```

La classe `Star` utilise les classes `Color` et `Graphics` du package `java.awt` et la classe `Raster` du package `java.awt.image`

Imports statiques

- Jusqu'à la version 1.4 de Java, pour utiliser un membre statique d'une classe, il faut obligatoirement préfixer ce membre par le nom de la classe qui le contient.

```
public class version1_4 {
  public static void main(String[] args) {
    System.out.println(Math.PI);
    System.out.println(Math.cos(0));
  }
}
```

- Java 1.5+ offre une nouvelle fonctionnalité pour réduire le code à écrire pour accéder aux membres statiques d'une classe : **l'import statique** (*static import*).
  - permet d'appliquer les mêmes règles aux membres statiques qu'aux classes et interfaces pour l'importation classique

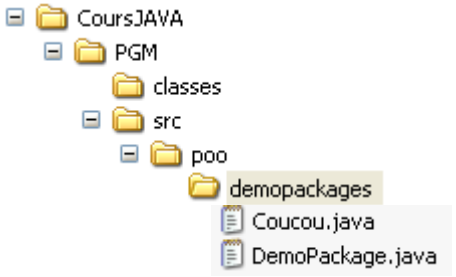
```
import static java.lang.Math.*;

public class version1_5{
  public static void main(String[] args) {
    System.out.println(PI);
    System.out.println(cos(0));
  }
}
```

importation statique s'applique à tous les membres statiques : constantes et méthodes statiques de l'élément importé

Compiler et exécuter

- Package **poo.demopackages**

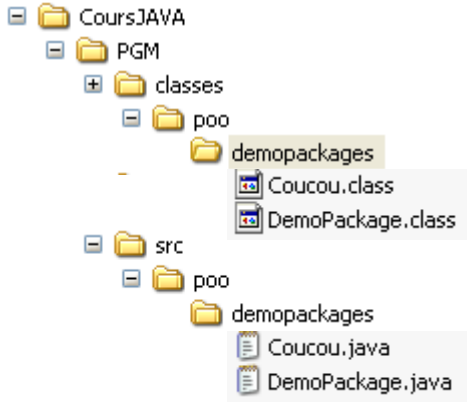
```
package poo.demopackages;

public class DemoPackage {

  public static void main(String[] args) {
     System.out.println("Demo packages");
     Coucou.hello();
  }
}
```



- Compiler
  - Se placer dans le répertoire racine des packages (**src**)
  - Spécifier tout le chemin pour désigner la classe

```
…\PGM\src>javac -d ../classes poo/demopackages/DemoPackage.java
```

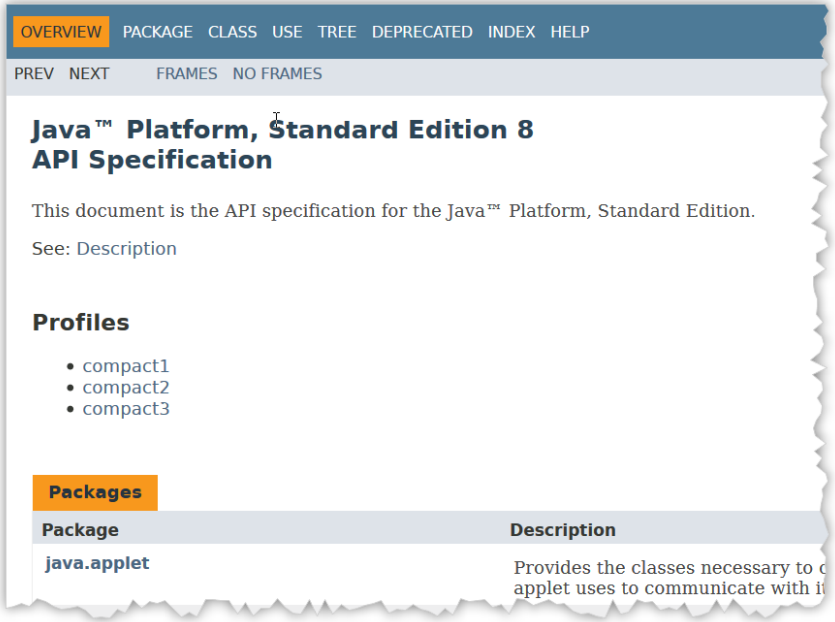
  - La hiérarchie des packages est reconstruite dans le répertoire de destination



- Exécuter
  - Se placer dans le répertoire racine des packages (**classes**)
  - Spécifier la classe en utilisant un nom complet (Fully qualified name)

```
…\PGM\classes>java poo.demopackages.DemoPackage
```

Profiles compacts de Java 8



Compact1 ⊂ Compact2 ⊂ Compact3

compact1	compact2	compact3
java.io	java.rmi	java.lang.instrument
java.lang	java.rmi.activation	java.lang.management
java.lang.annotation	java.rmi.registry	java.security.acl
java.lang.invoke	java.rmi.server	java.util.prefs
java.lang.ref	java.sql	javax.annotation.processing
java.lang.reflect	javax.rmi.ssl	javax.lang.model
java.math	javax.sql	javax.lang.model.element
java.net	javax.transaction	javax.lang.model.type
java.nio	javax.transaction.xa	javax.lang.model.util
java.nio.channels	javax.xml	javax.management
java.nio.channels.spi	javax.xml.datatype	javax.management.loading
java.nio.charset	javax.xml.namespace	javax.management.modelbean
java.nio.charset.spi	javax.xml.parsers	javax.management.monitor
java.nio.file	javax.xml.stream	javax.management.openmbean
java.nio.file.attribute	javax.xml.stream.events	javax.management.relation
java.nio.file.spi	javax.xml.stream.util	javax.management.remote
java.security	javax.xml.transform	javax.management.remote.rmi
java.security.cert	javax.xml.transform.dom	javax.management.timer
java.security.interfaces	javax.xml.transform.sax	javax.naming
java.security.spec	javax.xml.transform.stax	javax.naming.directory
java.text	javax.xml.transform.stream	javax.naming.event
java.text.spi	javax.xml.validation	javax.naming.ldap
java.util	javax.xml.xpath	javax.naming.spi
java.util.concurrent	org.w3c.dom	javax.script
java.util.concurrent.atomic	org.w3c.dom.bootstrap	javax.security.auth.kerberos
java.util.concurrent.locks	org.w3c.dom.events	javax.security.sasl
java.util.jar	org.w3c.dom.ls	javax.sql.rowset
java.util.logging	org.xml.sax	javax.sql.rowset.serial
java.util.regex	org.xml.sax.ext	javax.sql.rowset.spi
java.util.spi	org.xml.sax.helpers	javax.tools
java.util.zip		javax.xml.crypto
javax.crypto		javax.xml.crypto.dom
javax.crypto.interfaces		javax.xml.crypto.dsig
javax.crypto.spec		javax.xml.crypto.dsig.dom
javax.net		javax.xml.crypto.dsig.keyinfo
javax.net.ssl		javax.xml.crypto.dsig.spec
javax.security.auth		org.ieft.jgss
javax.security.auth.callback		
javax.security.auth.login		
javax.security.auth.spi		
javax.security.auth.x500		
javax.security.cert		

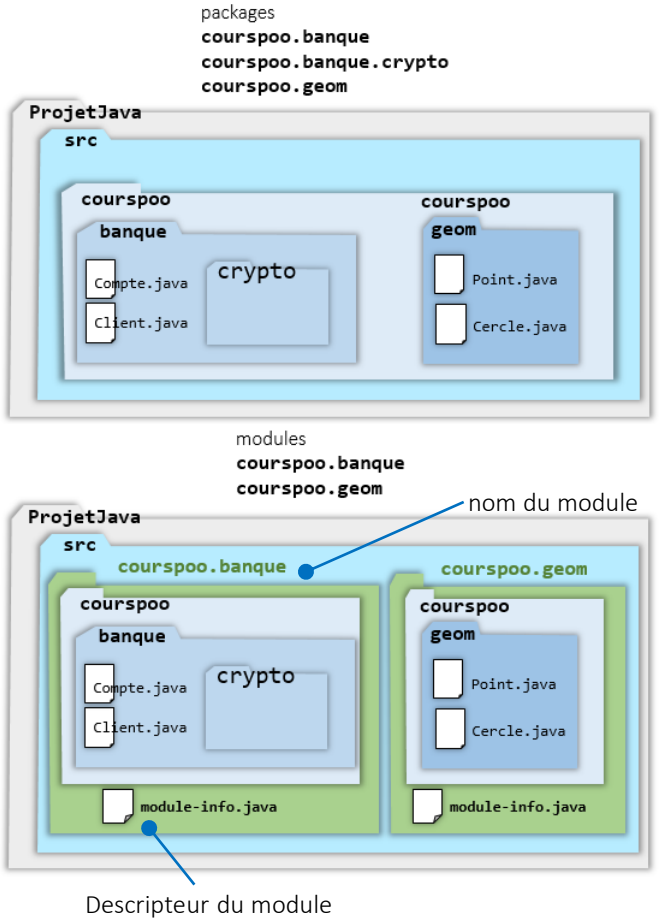
- Profiles compacts de Java8
  - Versions réduites du JRE (rt.jar)
  - Intérêts
    - Temps de démarrage de JVM plus rapide
    - Utilisation de moins de ressources
    - Suppression des paquets qui, avec le recul, ne devraient pas être dans le noyau
    - Meilleure sécurité
    - …

Une première étape vers la modularité de java introduite avec Java 9

Java 9+ : modules

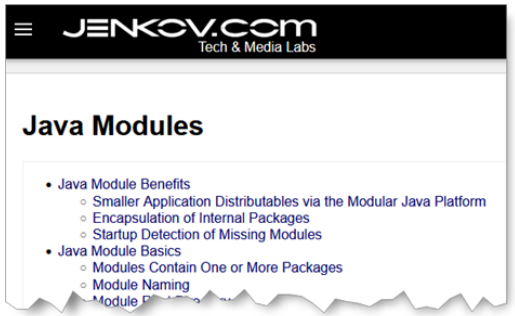
- Java Platform Module System (JPMS)
  - Nouveau niveau d'abstraction défini au dessus des packages

- Qu'est-ce qu'un module ?
  - Module : un groupe de packages et de ressources étroitement liés avec un nouveau fichier descripteur de module.
  - Une abstraction *d'un package de packages Java* afin de faciliter la réutilisabilité du code
  - Les packages dans un module sont identiques aux packages Java tels qu'ils sont définis dans les versions précédentes du langage.
  - Le descripteur de module permet de déterminer
    - quels autres modules sont nécessaires (dependencies)
    - quels codes (packages) sont accessibles en dehors du module (exports).



https://jenkov.com/tutorials/java/modules.html

Jakob Jenkov



Java 9+ : modules

- Descripteur de modules :
  - fichier définissant différentes propriétés du module
    - **name** : nom du module
      - Mêmes règles de nommage que pour les packages
    - **dependencies** : liste des autres modules dont dépend le module (ex : `my.module`)
    - **public packages** : package du module accessibles en dehors du module
      - Par défaut tous les packages d'un module sont privés
    - **services offered** : implémentations de services pouvant être consommés par d'autres modules
    - **services consumed** : permet au module d'être consommateur de services
    - **reflection permissions** : permet explicitement à d'autres classes d'utiliser l'API de reflection pour accéder aux membres privés d'un package
      - Par défaut on ne peut utiliser la reflection sur des classes importées d'un autre module
  - Nommé `module-info.java` et situé à la racine des packages

- Distribution
  - Sous la forme de fichiers jar : un module par `.jar`