

Import : exemples

- Applet

```
import java.applet.*;
import java.awt.*;
public class Salut extends Applet { //java.applet.Applet
    public void paint(Graphics g) { //java.awt.Graphics
        g.drawString("Salut!", 20, 20);
    }
}
```

- Collections

```
import java.util.*;
public class Bibliotheque {
    protected Map<String, Ouvrage> ouvrages
        = new TreeMap<String, Ouvrage>();
    ...
}
```

Création de packages

```
package <nomDePackage>;
```

- en tête de fichier, avant les imports éventuels
- « range » logiquement les classes du fichier dans le package
 - d'où le nom complet d'une classe (nécessaire si ambiguïté):


```
<nomDePackage>.<nomDeClasse>
java.util.HashMap, fr.polytech.ima4.circuit.MyClass
```
- Si aucun package n'est spécifié
 - les classes appartiennent à un package par défaut résidant sur le répertoire courant (.)
 - elles ne peuvent être importées ailleurs.
 - Une pratique à éviter
- Seules les classes déclarées public d'un package (autre que « . ») sont "import"-ables dans d'autres packages
 - Les autres sont encapsulées dans le package (classes auxiliaires d'implantation).

Exemple: accès aux classes

package lib : Ouvrage, Bibliotheque et Exception

```
//fichier ./lib/NonDisponibleException.java
```

```
package lib;
public class NonDisponibleException extends Exception{}
```

```
//fichier ./lib/Ouvrage.java
```

```
package lib;
public class Ouvrage { ...
    public void emprunter() throws NonDisponibleException ...
}
```

```
//fichier ./lib/Bibliotheque.java
```

```
package lib;
import java.util.*;
public class Bibliotheque {
    protected Map<String, Ouvrage> ouvrages ...
    public void emprunter(String code) throws NonDisponibleException
```

Exemple

- Utilisations des packages

```
//fichier ./elib/BibElectronique.java
```

```
package elib;
import lib.*;
public class BibElectronique extends Bibliotheque {...}
```

```
//fichier ./Application.java : sans package => package .
```

```
import lib.*;
import elib.*;
public class Application {
    public static void main(String[] argv) {
        Ouvrage x; //<=> lib.Ouvrage x;
        Bibliotheque bib = new Bibliotheque(); //lib.Bibliotheque
        BibElectronique ebib; //elib.BibElectronique
        ...
    }
}
```

Encapsulation

- régler le degré d'encapsulation/de visibilité entre classes
 - des variables d'instance : en général masquées pour cacher l'implantation (modularité)
 - des méthodes (et constructeurs) :
 - internes : accessoires d'implantation
 - publiques : protocole ou interface de manipulation

Visibilité entre classes	d'un même package	de packages distincts
public	oui	oui
protected (« subclass limited »)	oui	restreint aux sous-classes
Aucun / Default (« package limited »)	oui	non
private	non	non

Exemple: public/private

```
package complexes;
public class Complexe {
    private double re, im;
    public Complexe (double x, double y) {re=x; im=y;}
    public double re() {return re;}
    public Complexe add(Complexe c) {
        return new Complexe(re+c.re(),im+c.im());}
    ...
}
```

```
// meme package
// ou non (moyennant: import complexes.*)
public class Test {
    public static void main(String argv[]) {
        Complexe c = new Complexe(10.0,20.0); // public => ok
        ... c.re ... // erreur : private => non visible
        ... c.re= ... // encore moins!
        ... c.re() ... // public => ok
    }
}
```

Exemple: public/protected

```
package elib;
import lib.*;
public class BibElectronique extends Bibliotheque {
    public void listing(Display on) {
        for(Ouvrage o: ouvrages.values())
            // protected dans sous-classe OK
            on.show(ouvrages.get(code));
    }
}

import lib.*;
public class Application { //utilisatrice non sous-classe
    public static void main(String[] argv) {
        Bibliotheque bib = new Bibliotheque();
        bib.listing(); // public OK
        // bib.ouvrages.get(code).emprunte = false;
        // Impossible de tricher! : protected's hors sous-classe
    }
}
```

Quelques règles...

- Par défaut au sein d'un même package, tout est visible. Java part du principe qu'au sein d'un package on est entre amis (*friend* C++ :-)
- On ne peut redéfinir une méthode «plus privée» dans une sous-classe
 - Casse substituabilité
- Les modifieurs Java sont unitaires contrairement à C++ où l'on peut les faire porter sur un groupe de caractéristiques.
- La déclaration public d'une classe n'a aucune conséquence sur les modalités d'encapsulation de ses caractéristiques
- Le modifieur **final** appliqué à :
 - une classe : la rend non-extensible (ex. System)
 - une méthode : la rend non-redéfinissable
 - une variable initialisée : constante.
 - une pointeur : rends le pointeur non-modifiable
- Dans les documentations utilisateurs de classes (API Java ou les vôtres), seules les caractéristiques accessibles (public ou protected) apparaissent.