JAVA
FONDAMENTAUX

Présenté par :

Xavier TABUTEAU

Les packages et modules

En Java, l'organisation du code est essentielle pour la lisibilité, la réutilisation, et la maintenance.

Deux niveaux principaux permettent cette organisation :

Les packages : regroupent les classes, interfaces et sous-packages.
Les modules (introduits à partir de Java 9) : regroupent des packages et contrôlent leur accessibilité.

- **Les packages**

Un package est un espace de noms (namespace) qui sert à organiser les classes. Il permet d'éviter les conflits de noms entre classes, de structurer un projet en parties logiques, de contrôler la visibilité des classes. Lorsqu'on est dans une entreprise, la convention de nommage des packages dit qu'ils doivent commencer par l'inverse du site web de l'entreprise.

Exemple : Si l'entreprise possède un site nommé monentreprise.com alors les noms des packages fait pour cette entreprise doivent commencé par com.monentreprise.

Le package d'une classe se déclare sur la première ligne du fichier java que l'on crée avant tout autre code (sauf des commentaires éventuels). Le fichier doit se situé dans le dossier correspondant à l'arborescence décrite dans le package.

Exemple :

si l'emplacement du fichier est : src/com/monentreprise/utils/StringUtils.java
la déclaration du package dans le fichier est : package com.monentreprise.utils.StringUtils;

Les packages et modules

- **Utilisation des packages**

Pour pouvoir utiliser une classe d'un autre package, on doit faire un import.

Exemple :

```
import com.monentreprise.utils.StringUtils;
```

- **Java fournit de nombreux packages standards**

java.lang	Contient les classes de base (String, Math, Object, etc.).
java.util	Collections, dates, utilitaires.
java.io	Entrées/sorties de fichiers.
java.net	Programmation réseau.
java.sql	Connexion aux bases de données.

- **Contrôle d'accès avec les packages**

Modificateur	Accès dans le même package ?	Accès dans un autre package ?
public	oui	oui
protected	oui	oui (via héritage uniquement)
(sans modificateur)	oui	non
private	non	non

Les packages et modules

- **Les modules**

Un module est une unité plus grande qu'un package, introduite pour améliorer la modularité et la sécurité du code. Il regroupe plusieurs packages et définit ce qu'il exporte et ce dont il dépend. Chaque module Java est un sous-ensemble indépendant du projet, avec son propre répertoire source et un fichier « module-info.java » à la racine de ce module, et non du projet entier.

Exemple :

```
monProjet/
└── src/
    ├── com.monentreprise.utils/
    │   ├── module-info.java
    │   └── com/monentreprise/utils/StringUtils.java
    │
    └── com.monentreprise.app/
        ├── module-info.java
        └── com/monentreprise/app/Main.java
```

Les packages et modules

- **Les modules**

Contenu des fichiers modules :

```
module com.monentreprise.utils {
    exports com.monentreprise.utils;
}

module com.monentreprise.app {
    requires com.monentreprise.utils;
}
```

Le module app utilise (requires) le module utils.
Le module utils rend accessible (exports) son package com.monentreprise.utils

Avantages des modules :

Meilleure encapsulation : seuls les packages exportés sont accessibles.
Réduction des dépendances inutiles.
Démarrage plus rapide grâce à la modularisation de la JVM (Java Platform Module System - JPMS).
Sécurité et maintenance renforcées.

Les packages et modules

- **Les modules**

Lignes de commandes pour compiler et exécuter avec des modules :

Compilation : javac -d outDir --module-source-path src -m com.monentreprise.app
Exécution : java --module-path outDir -m com.monentreprise.app/com.monentreprise.app.Main

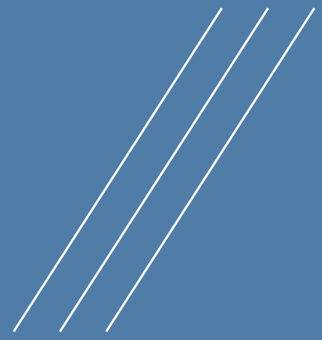
Résumé :

Les packages permettent depuis Java 1 de regrouper les classes et interfaces sans besoin de créer un fichier de configuration.
Les modules permettent de regrouper les packages depuis Java 9 mais nécessite de créer un fichier nommé « module-info.java » à la racine de chaque module.

Bonnes pratiques :

Utiliser des noms de packages uniques, basés sur un domaine inversé (ex : com.google.api).
Grouper logiquement les classes par fonctionnalité (ex : service, model, controller).
Ne pas trop multiplier les modules pour les petits projets.
Garder les modules cohérents et indépendants.

JAVA
FONDAMENTAUX



Présenté par
Xavier TABUTEAU