Chapitre 6
Les procédures et fonctions

1. Présentation

Les algorithmes réalisés avant ce chapitre utilisent ce qui se nomme le paradigme de programmation impérative. Un paradigme de programmation est une façon de programmer et il propose un certain nombre d'outils le permettant. Le paradigme de programmation impérative donne des ordres au travers des instructions qui se succèdent et propose différentes structures de contrôle permettant de sauter ou de refaire certaines parties en fonction de conditions.

Nous pouvons constater que cette manière de procéder a quelques limites. Les algorithmes deviennent de plus en plus longs au fur et à mesure que leur complexité augmente et il est de plus en plus difficile de s'y retrouver. De plus, il peut y avoir du code qui est dupliqué, car il est nécessaire de l'exécuter plusieurs fois à différents endroits de notre algorithme.

Dans ce chapitre, nous allons découvrir un deuxième paradigme de programmation. Il se nomme le paradigme de programmation procédurale. Il ne vient pas remplacer le précédent, mais il le complète.

Afin de pallier les limitations précédemment citées, le paradigme de programmation procédurale introduit une nouvelle notion : il s'agit des sous-algorithmes. Un algorithme peut faire appel à un sous-algorithme pour réaliser un travail. Prenons une comparaison pour bien comprendre. Un chef de service a un certain nombre d'objectifs à réaliser, il va demander à ses subalternes de réaliser des tâches et lui va coordonner le travail. Dans cette métaphore, le chef de service correspond à l'algorithme (nommé algorithme principal) et les subalternes correspondent aux sous-algorithmes.

L'utilisation de sous-algorithmes va permettre de répartir le code entre l'algorithme principal et des sous-algorithmes. Cela va structurer le code et le rendre plus lisible, mais également éviter la duplication du code.

Il existe deux types de sous-algorithmes : les procédures et les fonctions. Une procédure est un sous-algorithme contenant une suite d'instructions. Une fonction est similaire, à la différence qu'elle retourne, à la fin de son traitement, une valeur à l'algorithme qui y a fait appel. Pour reprendre la métaphore précédente, une procédure correspondrait par exemple à la tâche de traiter le courrier reçu. La seule chose qui est attendue en retour, c'est de savoir que la tâche est terminée. Une fonction pourrait être par exemple la tâche de calculer le montant des dernières commandes passées. Dans ce cas-là, il y a bien une valeur qui est attendue à la fin de la tâche.

Sans le savoir, vous utilisez déjà des procédures et des fonctions : `écrire()` est une procédure permettant l'affichage d'éléments sur la console et `saisir()` et `aléa()` sont des fonctions retournant une valeur, soit saisie par l'utilisateur, soit choisie aléatoirement.

2. La déclaration d'un sous-algorithme

La déclaration d'un sous-algorithme est très similaire à celle d'un algorithme.

2.1 Déclaration d'une procédure

Afin de déclarer une procédure, il faut utiliser la syntaxe suivante :

```
Procédure nom_de_la_procédure(liste_des_paramètres)
#Déclarations

Début
   #Instructions

Fin
```

Pour une procédure, seule la première ligne change. Tout d'abord, le mot-clé `Algo` est remplacé par le mot-clé `Procédure`. Ensuite, après le nom de la procédure, il y a un couple de parenthèses dans lesquelles il est possible de passer des paramètres. Les paramètres sont les informations nécessaires à la procédure pour qu'elle puisse réaliser sa tâche. La procédure `écrire()` prend par exemple en paramètre le texte à afficher sur la console. La liste de paramètres est une suite de couples `nom : type` séparés par des virgules. Les paramètres sont comme des variables qui sont initialisées lors de l'appel de la procédure.

Cette première ligne s'appelle la signature de la procédure.

Exemple de déclaration d'une procédure :

```
Procédure afficheNfois(t : texte, n : entier)
Variable i : entier
Début
 Pour i <- 1 à n
   écrire(t)
 FPour
Fin
```

Dans cet exemple, la procédure prend en paramètres deux valeurs : d'une part `t`, de type texte, le message à afficher, et d'autre part `n`, de type entier, le nombre de fois qu'il faut afficher ce message.

En Java, une procédure se déclare de la manière suivante :

```
public static void nom_de_la_procédure(liste_des_paramètres) {
 // traitements

}
```

Les mots-clés `public static void` sont expliqués plus loin dans cet ouvrage. Pour le moment, il faut retenir qu'il est nécessaire de les mettre pour déclarer une procédure. Vous avez sûrement remarqué que ces mots-clés vous sont familiers et que vous les utilisez pour coder votre algorithme principal :

```
public static void main(String[] args) {
   ...
}
```

Il s'agit en effet d'une procédure, c'est même celle qui se nomme la procédure principale. Elle se nomme forcément `main()` et prend en paramètre un tableau de chaînes de caractères.

Voici l'exemple de procédure précédent codé en Java :

```
public static void afficheNfois(String t, int n) {
   for (int i = 0; i < n; i++) {
         System.out.println(t);
   }
}
```

2.2 Déclaration d'une fonction

La déclaration d'une fonction est très proche de la déclaration d'une procédure. Voici la syntaxe de déclaration d'une fonction :

```
Fonction nom_de_la_fonction(listeParametres) Retourne type_retour
#Déclarations

Début
   #Instructions

   Retourner valeur
Fin
```

Tout d'abord, le mot-clé `Procédure` a bien entendu laissé place au mot-clé `Fonction`. Ensuite, après la liste des paramètres, un nouveau mot-clé fait son apparition : `Retourne`. Il est suivi du type de la valeur qui sera retournée par la fonction. La signature de la fonction permet ainsi de visualiser à la fois ce qui est attendu en paramètre, mais le type de valeur qui est retournée par celle-ci. Enfin, la dernière différence est juste avant le mot-clé `Fin`. Le mot-clé `Retourner` est suivi de la valeur qu'il faut retourner à l'algorithme ayant fait appel à cette fonction.

Exemple de déclaration d'une fonction :

```
Fonction puissance(a : réel, n : entier) Retourne réel
Variable i : entier
Variable p : réel <- 1
Début
   Pour i <- 1 à n
         p <- p * a
   FPour
   Retourner p
Fin
```

La fonction `puissance` prend en paramètres deux valeurs : d'une part la valeur que l'on souhaite élever à une puissance donnée et d'autre part l'exposant. Cette fonction calcule donc a^n, c'est-à-dire a à la puissance n.

Pour mémoire $a^n = a \times a \times \ldots \times a$ n fois.

Deux variables sont déclarées : la première, `i`, est la variable servant pour la boucle `Pour` et la seconde, `p`, est utilisée pour calculer la puissance. Initialement, cette variable vaut 1. Après la première itération de la boucle, elle vaut `1 × a`, c'est-à-dire `a`. À chaque itération, sa valeur est multipliée par `a`. Ainsi, après `n` itérations, cette variable vaut bien a^n. Au final, l'instruction `Retourner p` permet de retourner cette valeur à l'algorithme ayant fait appel à cette fonction.

En Java, la syntaxe de déclaration d'une fonction est la suivante :

```
public static type_retour
       nom_de_la_fonction(liste_des_paramètres) {
 // traitements

 return valeur;
}
```

Il n'y a pas de mot-clé particulier pour indiquer qu'il s'agit d'une fonction, mais le mot-clé `void` a été remplacé par le type de retour de la fonction. Une procédure est en fait en Java un cas particulier de fonction qui ne renvoie rien (du vide !). Le mot-clé `return` permet de mettre fin à la fonction et de retourner la valeur souhaitée.

Voici la fonction puissance codée en Java :

```
public static double puissance(double a, int n) {
   double p = 1;
   for(int i=0; i<n; i++)
         p *= a;
   return p;
}
```

Cette fonction déclare qu'elle retourne un type `double`, c'est-à-dire un nombre réel. Elle prend en paramètres deux valeurs : un nombre réel et un nombre entier. À la fin de la fonction, le résultat de calcul de la puissance est stocké dans la variable p et c'est cette valeur qui est retournée.

3. L'appel à un sous-algorithme

3.1 L'appel à une procédure

Pour faire appel à une procédure, il faut simplement écrire le nom de la procédure suivie, entre parenthèses, des valeurs permettant d'initialiser les paramètres dans l'ordre dans lequel ceux-ci sont attendus.

Exemple :

```
Algo Punition
# demande à l'utilisateur le texte à copier et le nombre de fois
# qu'il faut le copier
Variable  phrase : texte
Variable  nbFois : entier
Début
 phrase <- saisir("Quelle phrase avez-vous à copier comme punition ?")
 nbFois <- saisir("Combien de fois avez-vous eu à la copier ?")
 afficheNfois(phrase, nbFois)
Fin
```

Chapitre 2
Prise en main d'Eclipse

1. De l'importance de s'organiser

Ce chapitre permet de se familiariser avec l'environnement de développement Eclipse.

Créer un logiciel implique de modifier des fichiers texte. Au fur et à mesure de cette création, les fichiers s'accumulent et il devient de plus en plus important de s'organiser afin de pouvoir retrouver facilement le code.

Pour cela, Java propose un système d'organisation par espace de noms ou package, qui permet de regrouper différentes classes. Conceptuellement, il est plus simple d'y penser comme à différents dossiers de l'ordinateur.

Au plus haut niveau d'organisation, le code d'un logiciel est organisé dans un projet par Eclipse. Le projet regroupe tous les codes nécessaires à la réalisation des fonctionnalités demandées. Chaque élément de code est ensuite regroupé dans des dossiers différents selon qu'ils soient utiles à la fonctionnalité proprement dite ou à leurs tests.

2. Premier projet

Pour se familiariser avec le **workbench**, une classe très simple est créée.

Il faut auparavant créer un projet puis un package, toute classe d'une application un tant soit peu professionnelle devant être rangée dans un package.

Pour rappel, un package est physiquement un dossier, et symboliquement un espace de nommage pour rassembler les classes d'une même fonctionnalité.

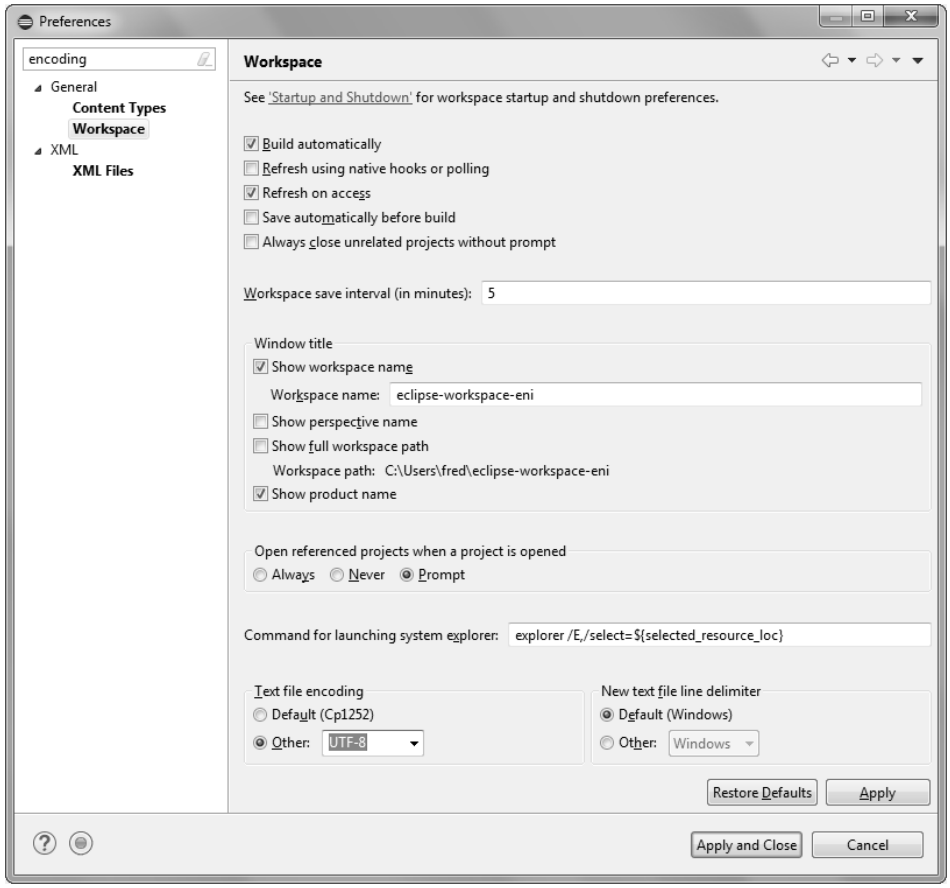
Remarque

Si vous ne rangez pas vos classes dans un package, Eclipse les placera dans le package par défaut. L'utilisation du package par défaut est déconseillée.

C'est également l'occasion de proposer une structure pour les fichiers du projet, afin d'améliorer la lisibilité et la testabilité des applications.

En préambule à ce travail, et afin de faciliter l'ouverture des fichiers provenant de systèmes d'exploitation différents, il convient de préciser l'encodage des fichiers.

- Dans le menu, choisissez **Window - Preferences**. Naviguez ensuite dans **General - Workspace** et sélectionnez la valeur **Other - UTF-8** dans la section **Text file encoding**.

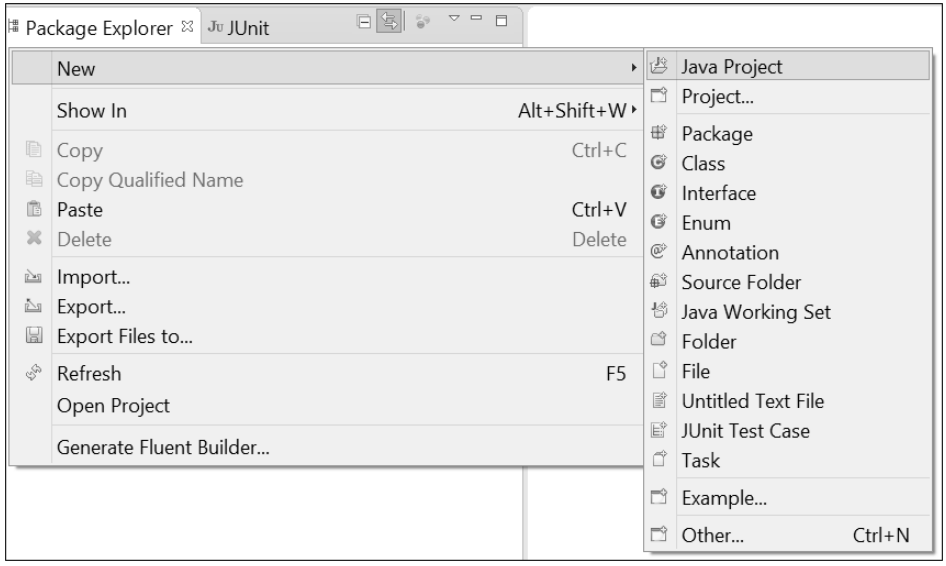


Ce réglage permettra de lire correctement les fichiers avec des caractères accentués dans Eclipse, que ce soit sur Windows, Linux ou Mac.

Remarque

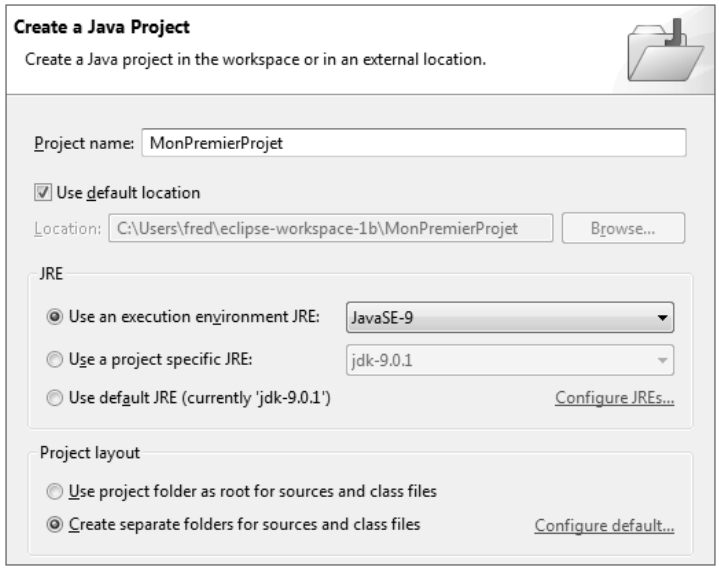
Les accents ainsi que les caractères spéciaux sont permis dans les noms des variables, mais ne sont pas recommandés. Vous comprendrez le jour où votre projet sera ouvert sur un ordinateur Linux, par exemple, ou quand il s'agira de travailler sur un projet avec les noms de variables en chinois, japonais, hébreu ou farsi.

- Dans le menu, choisissez **File - New - Java Project**, ou faites un clic droit dans la vue puis **New - Java Project**.



La boîte de dialogue de création de projet s'ouvre. Elle permet de nommer le nouveau projet et de choisir l'emplacement sur le disque dur où seront stockés les fichiers du projet (par défaut, les projets sont stockés dans le dossier d'espace de travail qui a été choisi lors de l'installation d'Eclipse).

- Saisissez le nom du projet, puis cliquez sur le lien **Configure default** en face du bouton radio **Create separate folders for sources and class files**.



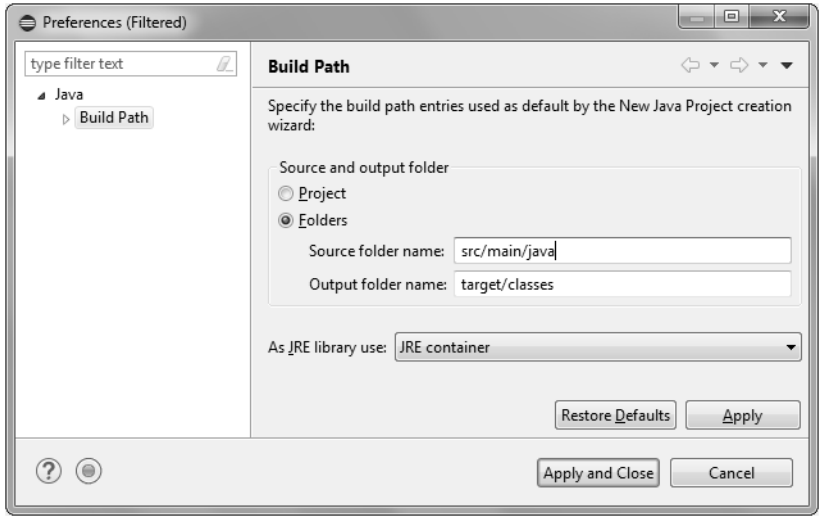
Cette manipulation permet de définir les dossiers dans lesquels sont rangés les fichiers sources et les fichiers binaires compilés. D'une manière générale, ces deux types de fichiers doivent être séparés. Cela permet par exemple de copier ou d'effacer facilement les binaires.

Les fichiers sources dans un projet Java ont souvent plusieurs sous-types : il existe des fichiers sources pour les classes de l'application, des fichiers sources pour les classes de tests, des fichiers de ressources telles que les images ou les configurations…

La manipulation suivante permet de définir des dossiers distincts pour chacun de ces types. Ces dossiers distincts sont inspirés des projets Maven (http://maven.apache.org/) et Gradle (https://gradle.org/), qui sont des outils puissants pour la gestion des projets informatiques, permettant de lancer en ligne de commande ou via une interface graphique la compilation des sources en binaires, le passage des tests unitaires, la transformation des classes en fichier jar, le paquetage dans un installateur et le déploiement sur un serveur, pour ne citer que quelques-unes des fonctionnalités disponibles.

Maven met en application l'adage « Une place pour chaque chose et chaque chose à sa place » et définit le répertoire **src/main/java** comme répertoire pour les fichiers sources, **src/main/resources** pour les fichiers de configuration et les images, **src/test/java** pour les sources de tests et **src/test/resources** pour les fichiers de configuration des tests. Les binaires sont stockés quant à eux dans **target/classes**.

▶Changez les noms des dossiers comme dans l'écran suivant, puis validez.



Eclipse retourne sur la boîte de dialogue de création du projet.

▶Cliquez sur **Finish** pour finaliser la création du projet.

Le projet et son contenu sont maintenant visibles dans la vue **Package Explorer**.