Nathan Guhéneuf-Le Brec
Jean-Louis Emeraud
Iñaki Gomez—Jego
François Patinec-Haxel

Introduction

Dans le cadre de la section « Programmation Orientée Objets » de la SAE Communes Bretonnes, nous avons été amenés à programmer les différentes classes de la partie modèle de notre application.

Tout d'abord, nous avons traduit les classes du diagramme de conception de la base de données fourni sous forme de classes java puis nous y avons rajouté des méthodes qui travaillent en symbiose avec plusieurs classes que nous avons créées afin d'obtenir une liste triée selon un filtre que l'utilisateur choisira dans la partie vue de notre application.
Ces classes se trouvent dans le paquet modele.classesModele et plus de détails quant à la traduction de celles-ci et quant aux méthodes ajoutées sont disponibles dans la section « Paquet modele.classesModele » de ce document.

Après cela, nous avons créé des classes qui travaillent en symbiose avec les méthodes que nous avons ajoutées aux classes du paquet modele.classesModele.
Celles-ci peuvent nottament effectuer plusieurs types de méthodes de recherches et plusieurs types de méthodes de tri sur une liste d'objets de même type à l'aide de filtres fournis par l'utilisateur.
Ces classes se trouvent dans le paquet modele.methodesTri et plus de détails quant à la motivation derrière leur création et quant à leur utilité sont disponibles dans la section « Paquet modele.methodesTri » de ce document.

Enfin, nous avons créer une classe Scenario.java qui permet de tester toutes les méthodes des deux paquets exceptées les méthodes de recherche qui seront utilisées une fois que notre contrôleur est complété.
Cette classe est directement stockée dans le dossier src, plus de détails sur celle-ci sont disponibles dans la section « Notre scénario » de ce document.

Ci-dessous se trouve un sommaire vous permettant de vous déplacer vers les différentes sections de ce document.

I. Paquet modele.classesModele

1. Traduction

Lors de la traduction des classes du diagramme de conception de la base de données vers des classes java, nous avons choisi de garder les noms et les types proposés par celui-ci pour nos attributs.
De plus, nous avons remplacé les clés étrangères par des attributs associés à la ou les classes correspondantes. Par exemple, la classe Gare contient l'attribut laCommune de type Commune puisque chaque instance de la classe Gare doit être rattachée à une commune existante c'est-à-dire à une instance de la classe Commune.
Enfin, nous avons choisi de traduire la contrainte sur la classe Departement en utilisant un attribut que nous avons appelé DEPARTEMENTS_AUTORISES.
Cet attribut est à la fois static, final et est de type List<String> afin qu'il ne soit pas modifiable et afin qu'il soit utilisé dans la méthode departementEstCorrect(String nomDep) qui renvoie true si et seulement si l'argument nomDep passé en paramètre est contenu dans cette liste.
Puisque cette méthode est appelée dès qu'on modifie le nom d'un département, il est impossible de rentrer un nom qui ne respecte pas la contrainte et donc celle-ci est nécessairement respectée.

2. Méthodes ajoutées

Une fois que nous avons traduit nos classes java, nous avons décidé d'ajouter les attributs statiques currentFilter et FILTER_LIST, dont le contenu consiste en le nom des attributs de la classe sous forme de chaînes de caractères, ainsi que les méthodes statiques getAllFilter() et setFilter(String filter) à toutes les classes afin de pouvoir manipuler le contenu textuel de l'attribut statique currentFilter.
Après, nous avons fait implémenter l'interface Comparable<T> à toutes les classes afin qu'elles contiennent leur implémentation de la méthode compareTo(T o) où nous utilisons le contenu textuel de l'attribut statique currentFilter afin de déterminer, selon la valeur de celui-ci, lequel des attributs de la classe nous devons utiliser pour effectuer la comparaison entre deux objets.
Cette implémentation de la méthode compareTo(T o) est dans l'optique de l'utilisation des résultats qu'elle renvoie pour les méthodes de tri du paquet modele.methodesTri afin de trier une liste d'objets de même type dans un ordre croissant selon un paramètre bien précis de ceux-ci.

II. Paquet modele.methodesTri

1. Motivations derrière la création des classes du paquet

La motivation derrière la création des classes de ce paquet est simple.
Nous cherchons à donner à l'utilisateur de notre application la capacité de manipuler les données qui lui sont fournies ainsi qu'à lui donner une capacité de visualisation de celles-ci à travers une importante liberté de manipulation des actions effectuées par l'application.
Pour cela, nous avons besoin de méthodes de recherche efficaces qui seront utilisées lors de la modification des données (recherche d'un objet puis modification d'un attribut de celui-ci) ainsi que des méthodes de tri efficaces qui seront utilisées afin de répondre à différents besoins de l'utilisateur tels que pouvoir, par exemple, visualiser les communes de Bretagne sous forme d'une liste triée dans l'ordre décroissant de leur population.
Puisque que ces besoins sont difficillement prévisibles, nous avons décidé d'implémenter les attributs statiques et méthodes statiques mentionnées dans la section précédente de façon à fournir une liberté de manipulation des actions de l'application à la fois efficace et satisfaisante.

2. Utilité des classes du paquet

Tout d'abord, le paquet contient deux interfaces. La première, ISearch<T>, contient la signature des méthodes boolean exists(ArrayList<T> arr, T obj) et int search(ArrayList<T> arr, T obj). La deuxième, ITri<T>, contient la signature de la méthode void trier().

Dans chaque implémentation de l'interface ISearch<T>, la méthode exists sera utilisée pour vérifier si un élément donné existe dans une liste triée et la méthode search sera utilisée pour trouver l'indice d'un élément donné dans une liste triée.
Dans chaque implémentation de l'interface ITri<T>, la méthode trier sera utilisée pour lancer l'algorithme de tri utilisé pour trier une collection.

Les classes dont la signature contient le mot-clé « Searcher » sont des classes utilisées lors de la recherche d'un objet en particulier dans une liste d'objets de même type.
Celles-ci implémentent l'interface ISearch<T> et redéfinissent les méthodes boolean exists(ArrayList<T> arr, T obj) et int search(ArrayList<T> arr, T obj) de façon à correspondre à l'utilisation de la classe.
Par exemple, la classe BinarySearcher consiste en une implémentation de l'algorithme de la recherche dichotomique ainsi les méthodes exists et search contiennent le code nécessaire à cette implémentation.

Les autres classes du paquet consistent en des classes implémentant les algorithmes du tri rapide et du tri par sélection.
Celles-ci implémentent l'interface ITri<T> et redéfinissent la méthode trier de façon à lancer l'algorithme de tri correspondant à celui indiqué par le nom de la classe.

III. Notre scénario

Notre classe Scenario.java contient une méthode public static void main(String[] args) qui lance les méthodes de test de la classe.
Celle-ci contient deux méthodes de test, testTri() et triPourDiagrammeDeSequence(). Les autres méthodes de la classe, c'est-à-dire initialization(), getCurrentFilter(int i) et useCurrentFilter(int i, String filter, Object[] listeInstances) sont utilisées poru factoriser le code.

La première méthode de test consiste en la création de listes d'objets pour toutes les classes programmées dans le paquet modele.classesModele sur lesquelles ont effectue un tri rapide pour tous les filtres disponibles de chaque classe puis on imprime le résultat dans le terminal afin de pouvoir visualiser si celui-ci est le résultat attendu en fonction du filtre choisi.

La deuxième méthode de test consiste en la création d'une liste d'objets de type Departement[] contenant trois objets Departement sur lesquelles on utilise un tri par sélection afin de trier cette liste d'objets. De plus, cette méthode correspond à celle dont l'on visualise le déroulement dans notre diagramme de séquence.