Cohésion

Nous avons introduit précédemment (section « Introduction au couplage et à la cohésion ») le concept de cohésion : une partie de code devrait toujours être chargée d'une tâche et une seule. Nous allons maintenant étudier le principe de cohésion plus en profondeur et analyser quelques exemples.

Le principe de cohésion peut s'appliquer à des méthodes ou à des classes : classes et méthodes devraient fournir un haut degré de cohésion.

Cohésion des méthodes

Quand nous parlons de cohésion des méthodes, nous cherchons à exprimer la situation idéale où toute méthode devrait être responsable d'une tâche précise.

Concept

Une **méthode cohésive** est responsable d'une tâche précise.

La classe `Game` contient un exemple de méthode cohésive. Cette classe possède une méthode privée nommée `printWelcome` pour afficher le texte d'accueil et cette méthode est appelée quand la partie débute dans la méthode `play` (code 7.8).

Code 7.8 • Deux méthodes possédant un bon degré de cohésion.

```
    /**
     * Fonction principale de jeu. Boucle jusqu'à la
     * fin du jeu.
     */
    public void play()
    {
        printWelcome();

        // Entrée dans la boucle principale des commandes.
        // Ici, nous répétons la lecture et l'exécution des
        // commandes jusqu'à la fin du jeu.

        boolean finished = false;
        while (! finished) {
            Command command = parser.getCommand();
```

```
            finished = processCommand(command);
        }
        System.out.println("Merci d'avoir joué.
                Au revoir.");
    }

    /**
     * Affichage du message d'accueil au joueur.
     */
    private void printWelcome()
    {
        System.out.println();
        System.out.println("Bienvenue au jeu de Zuul !");
        System.out.println("Zuul est un nouveau jeu
           d'aventure terriblement ennuyeux.");
        System.out.println("Tapez 'aide' si vous avez besoin
           d'aide.");
        System.out.println();
        System.out.println(currentRoom.getLongDescription());
    }
```

D'un point de vue fonctionnel, nous aurions pu simplement ajouter les instructions de la méthode `printWelcome` à la méthode `play` et obtenir le même résultat sans définir une nouvelle méthode. Nous pouvons dire la même chose de la méthode `processCommand`, également invoquée par la méthode `play`.

Il est cependant plus facile de comprendre ce que fait une partie de code, et de la modifier, quand les méthodes utilisées sont courtes et cohésives. Dans la structure retenue, chaque méthode est raisonnablement courte et facile à comprendre, et son nom précise assez clairement son rôle. Ces caractéristiques représentent une aide précieuse pour un programmeur chargé de la maintenance.

Cohésion des classes

La règle de cohésion des classes signifie que chaque classe doit représenter une entité précise dans le domaine du problème.

Concept

Cohésion des classes. Une classe cohésive représente une entité précise.

Comme exemple de cohésion de classe, nous allons discuter une autre extension du projet *zuul*. Nous voulons maintenant ajouter des objets au jeu. Chaque pièce peut contenir un objet, et chaque objet possède une description et un poids. Le poids d'un objet peut être utilisé pour déterminer s'il peut être emporté ou non.

Une solution naïve consiste à ajouter deux champs à la classe `Room` : `itemDescription` et `itemWeight`. Cela peut fonctionner. Nous pouvons maintenant préciser les détails de chaque objet pour chaque pièce et nous pouvons les afficher chaque fois que nous pénétrons dans une pièce.

Cependant, cette approche ne présente pas un bon degré de cohésion : la classe `Room` décrit maintenant une pièce et un objet. Elle suggère également qu'un objet appartient à une seule pièce, ce que nous pouvons ne pas souhaiter.

Une meilleure conception crée une classe distincte pour les objets, probablement appelée `Items`. Cette classe possède des champs pour stocker la description et le poids, et une pièce dispose simplement d'une référence vers un objet de type `Item`.