nous écrivons maintenant :

```
nextRoom = currentRoom.getExit('est');
```

Cette modification simplifie en outre une partie de la classe Game. Dans la méthode goRoom, le remplacement suggéré ici aboutit à la partie de code suivante :

```
Room nextRoom = null;
if(direction.equals('nord')) {
    nextRoom = currentRoom.getExit("nord");
}
if(direction.equals('est')) {
    nextRoom = currentRoom.getExit("est");
}
if(direction.equals("sud")) {
    nextRoom = currentRoom.getExit('sud");
}
if(direction.equals("ouest")) {
    nextRoom = currentRoom.getExit("ouest');
}
```

Cette partie entière de code peut maintenant être remplacée par :

```
Room nextRoom = currentRoom.getExit(direction);
```

Exercice 7.6

Apportez les modifications décrites aux classes Room et Game.

Exercice 7.7

Modifiez de façon similaire la méthode printLocationInfo de la classe Game afin que les détails des sorties soient préparés par la classe Room plutôt que par la classe Game. Dans la classe Room, définissez une méthode dont voici la signature :

```
/**
 * Renvoie une description des sorties de la
 * pièce, par exemple, "Sorties : nord ouest".
 * @return Une description des sorties possibles.
 */
public String getExitString()
```

Code 7.5 • Code source de la classe Room.

```
import java.util.HashMap;

// omission des commentaires de la classe

public class Room
{
    private String description;
    private HashMap<String, Room> exits;

     /**
     * Crée une pièce décrite par la chaîne 'description'
     * Au départ, il n'existe aucune sortie.
     * "description" est une chaîne comme 'une cuisine' ou
     * 'une cour de jardin'.
     */
    public Room(String description)
    {
        this.description = description;
        exits = new HashMap<String, Room>();
    }

    /**
     * Définit les sorties de cette pièce. Chaque direction
     * soit conduit à une autre pièce, soit est null (il n'y
     * a pas de sortie dans cette direction).
     */
   public void setExits(Room north, Room east, Room south,
                        Room west)
   {
      if(north !=null)
          exits.put("nord", north);
      if(east !=null)
          exits.put("est", east);
      if(south !=null)
          exits.put('sud', south);
      if(west !=null)
          exits.put('ouest", west);
   }

   /**
    * Renvoie la pièce atteinte si nous nous déplaçons
    * dans la direction "direction". S'il n'y a pas de pièce
    * dans cette direction, renvoie null.
    */
   public Room getExit(String direction)
   {
       return exits.get(direction);
   }
```

```
    /**
     * Renvoie la description de la pièce
     * (telle que définie par le constructeur).
     */
    public String getDescription()
    {
        return description;
    }
}
```

Insistons encore sur le fait que nous pouvons maintenant faire ce changement sans même contrôler si une erreur peut se produire ailleurs. Comme nous n'avons modifié que des aspects privés de la classe `Room` qui, par définition, ne peuvent être utilisés par les autres classes, ces changements n'ont aucune influence sur les autres classes. L'interface reste inchangée.

Effet secondaire de cette modification, notre classe `Room` est maintenant plus petite. Au lieu d'énumérer quatre variables distinctes, nous n'en avons plus qu'une. De plus, la méthode `getExit` est considérablement simplifiée.

Rappelons-nous que l'objectif qui a motivé ces modifications est de faciliter l'ajout de deux nouvelles directions possibles vers le *haut* et le *bas*. Cette tâche est déjà plus facile à accomplir. Comme nous utilisons maintenant un objet de type `HashMap` pour mémoriser les sorties, stocker les deux directions supplémentaires fonctionnera sans modification. Nous pouvons aussi obtenir les informations de sortie par la méthode `getExit` sans problème.

Le seul endroit où la connaissance des quatre sorties existantes (*nord*, *est*, *sud*, *ouest*) est toujours utilisée dans le code source est la méthode `setExits`. C'est la dernière partie où des modifications sont nécessaires. Pour l'instant, la signature de la méthode est :

```
public void setExits(Room north, Room east, Room south, Room west)
```

Cette méthode fait partie de l'interface de la classe `Room`. Ainsi, toute modification que nous lui apporterons aura inévitablement des répercussions sur d'autres classes du fait du couplage. Il est important de noter que nous ne pouvons jamais totalement découpler les classes d'une application ; sinon, des objets de classes différentes ne pourraient pas interagir. Par contre, nous essayons de garder un degré de couplage aussi faible que possible. Si nous devons quand même modifier la méthode `setExits` pour faciliter la prise en compte de directions supplémentaires, notre solution privilégiée consiste à remplacer cette méthode par la suivante :

```
    /**
     * Définit une sortie pour cette pièce.
     * @param direction La direction de la sortie.
     * @param neighbor La salle dans la direction donnée.
     */
    public void setExit(String direction, Room neighbor)
    {
        exits.put(direction, neighbor);
    }
```

Les sorties de cette pièce peuvent maintenant être définies une par une, et n'importe quelle direction peut être utilisée comme sortie. Ce changement dans l'interface de la classe `Room` aboutit à la modification suivante de la classe `Game`. Au lieu d'écrire :

```
lab.setExits(outside, office, null, null);
```

nous écrivons maintenant :

```
lab.setExit("nord", outside);
lab.setExit("est", office);
```

Nous avons maintenant complètement supprimé la restriction concernant le stockage de quatre directions seulement par la classe `Room`. Cette classe est à présent en mesure de stocker les sorties *haut* et *bas*, ainsi que toute autre direction à laquelle vous pourriez penser (nord-ouest, sud-est, etc.).

Exercice 7.8

Implantez les modifications de cette partie dans votre propre projet *zuul*.