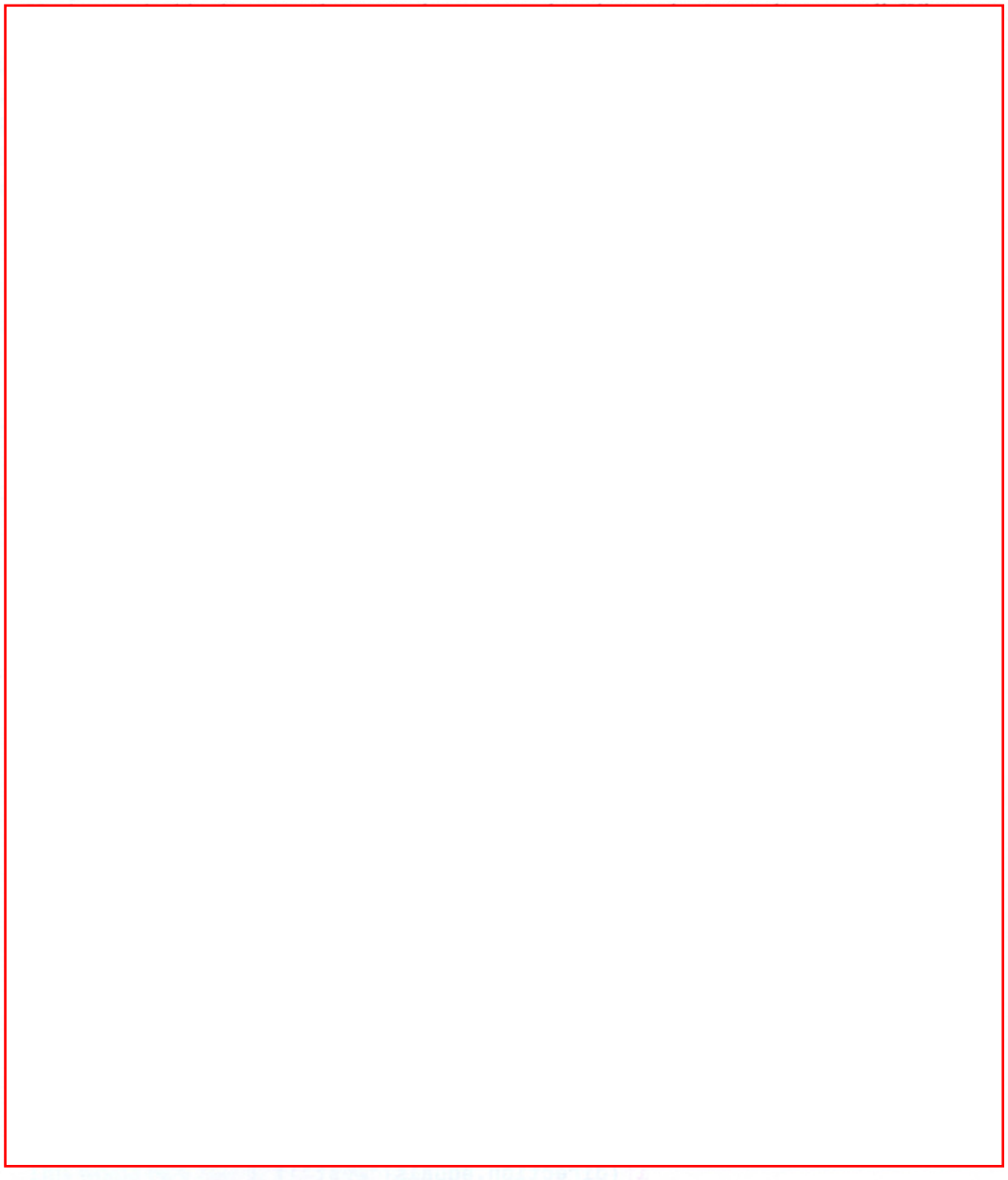

So far, we have not changed the representation of the exits in the `Room` class. We have only cleaned up the interface. The *change* in the `Game` class is minimal – instead of an access of a public field, we use a method call – but the *gain* is dramatic. We can now make a change to the way exits are stored in the room, without any need to worry about breaking anything in the `Game` class. The internal representation in `Room` has been completely decoupled from the interface. Now that the design is the way it should have been in the first place, exchanging the separate exit fields for a `HashMap` is easy. The changed code is shown in Code 7.5.

Code 7.5
Source code of the Room class

```
import java.util.HashMap;

// class comment omitted

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

    /**
     * Create a room described "description". Initially, it
     * has no exits. "description" is something like "a
     * kitchen" or "an open court yard".
     */
    public Room(String description)
    {
        this.description = description;
        exits = new HashMap<String, Room>();
    }

    /**
     * Define the exits of this room. Every direction either
     * leads to another room or is null (no exit there).
     */
    public void setExits(Room north, Room east, Room south,
                         Room west)
    {
        if(north != null)
            exits.put("north", north);
        if(east != null)
            exits.put("east", east);
        if(south != null)
            exits.put("south", south);
        if(west != null)
            exits.put("west", west);
    }

    /**
     * Return the room that is reached if we go from this
     * room in direction "direction". If there is no room in
     * that direction, return null.
     */
    public Room getExit(String direction)
    {
        return exits.get(direction);
    }
```

Code 7.5 continued
Source code of the `Room` class

```
    /**
     * Return the description of the room (the one that was
     * defined in the constructor).
     */
    public String getDescription()
    {
        return description;
    }
}
```

It is worth emphasizing again that we can make this change now without even checking whether anything will break elsewhere. Since we have changed only private aspects of the `Room` class, which, by definition, cannot be used in other classes, this change does not impact on other classes. The interface remains unchanged.

A by-product of this change is that our `Room` class is now even shorter. Instead of listing four separate variables, we have only one. In addition the `getExit` method is considerably simplified.

Recall that the original aim that set off this series of changes was to make it easier to add the two new possible exits in the *up* and *down* direction. This has already become a lot easier. Since we now use a `HashMap` to store exits, storing these two additional directions will work without any change. We can also obtain the exit information via the `getExit` method without any problem.

The only place where knowledge about the four existing exits (*north, east, south, west*) is still coded into the source is in the `setExits` method. This is the last part that needs improvement. At the moment, the method's signature is

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

This method is part of the interface of the `Room` class, so any change we make to it will inevitably affect some other classes by virtue of coupling. It is worth noting that we can never completely decouple the classes in an application, otherwise objects of different classes would not be able to interact with one another. Rather we try to keep the degree of coupling as low as possible. If we have to make a change to `setExits` anyway, to accommodate additional directions, then our preferred solution is to replace it entirely with this method:

```
/**
 * Define an exit from this room.
 * @param direction The direction of the exit.
 * @param neighbor The room in the given direction.
 */
public void setExit(String direction, Room neighbor)
{
    exits.put(direction, neighbor);
}
```

Now, the exits of this room can be set one exit at a time, and any direction can be used for an exit. In the `Game` class, the change that results from modifying the interface of `Room` is as follows. Instead of writing

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

we now write

```
lab.setExit("north", outside);
lab.setExit("east", office);
```

We have now completely removed the restriction from `Room` that it can store only four exits. The `Room` class is now ready to store *up* and *down* exits, as well as any other direction you might think of (northwest, southeast, etc.).

Exercise 7.8 Implement the changes described in this section in your own *zuul* project.