7.4 Code duplication

Concept:

Code duplication (having the same segment of code in an application more than once) is a sign of bad design. It should be avoided.

Code duplication is an indicator of bad design. The `Game` class shown in Code 7.1 contains a case of code duplication. The problem with code duplication is that any change to one version must also be made to the other if we are to avoid inconsistency. This increases the amount of work a maintenance programmer has to do, and it introduces the danger of bugs. It happens very easily that a maintenance programmer finds one copy of the code and, having changed it, assumes that the job is done. There is nothing indicating that a second copy of the code exists, and it might incorrectly remain unchanged.

Code 7.1
Selected sections of the (badly designed) `Game` class

```
public class Game
{
    // ... some code omitted ...

    private void createRooms()
    {
        Room outside, theatre, pub, lab, office;

        // create the rooms
        outside = new Room(
                  "outside the main entrance of the university");
        theatre = new Room("in a lecture theatre");
        pub = new Room("in the campus pub");
        lab = new Room("in a computing lab");
        office = new Room("in the computing admin office");

        // initialise room exits
        outside.setExits(null, theatre, lab, pub);
        theatre.setExits(null, null, null, outside);
        pub.setExits(null, outside, null, null);
        lab.setExits(outside, office, null, null);
        office.setExits(null, null, null, lab);

        currentRoom = outside; // start game outside
    }

    // ... some code omitted ...
```

Code 7.1 continued
Selected sections of the (badly designed) Game class

```
    /**
     * Print out the opening message for the player.
     */
    private void printWelcome()
    {
        System.out.println();
        System.out.println("Welcome to the World of Zuul!");
        System.out.println(
            "Zuul is a new, incredibly boring adventure game.");
        System.out.println("Type 'help' if you need help.");
        System.out.println();
        System.out.println("You are " +
                           currentRoom.getDescription());
        System.out.print("Exits: ");
        if(currentRoom.northExit != null) {
            System.out.print("north ");
        }
        if(currentRoom.eastExit != null) {
            System.out.print("east ");
        }
        if(currentRoom.southExit != null) {
            System.out.print("south ");
        }
        if(currentRoom.westExit != null) {
            System.out.print("west ");
        }
        System.out.println();
    }

    // ... some code omitted ...

    /**
     * Try to go to one direction. If there is an exit, enter
     * the new room, otherwise print an error message.
     */
    private void goRoom(Command command)
    {
        if(!command.hasSecondWord()) {
            // if there is no second word,
            // we don't know where to go
            System.out.println("Go where?");
            return;
        }
```

Code 7.1 continued
Selected sections of the (badly designed) `Game` class

```
        String direction = command.getSecondWord();

        // Try to leave current room.
        Room nextRoom = null;
        if(direction.equals("north")) {
            nextRoom = currentRoom.northExit;
        }
        if(direction.equals("east")) {
            nextRoom = currentRoom.eastExit;
        }
        if(direction.equals("south")) {
            nextRoom = currentRoom.southExit;
        }
        if(direction.equals("west")) {
            nextRoom = currentRoom.westExit;
        }
        if(nextRoom == null) {
            System.out.println("There is no door!");
        }
        else {
            currentRoom = nextRoom;
            System.out.println("You are " +
                               currentRoom.getDescription());
            System.out.print("Exits: ");
            if(currentRoom.northExit != null) {
                System.out.print("north ");
            }
            if(currentRoom.eastExit != null) {
                System.out.print("east ");
            }
            if(currentRoom.southExit != null) {
                System.out.print("south ");
            }
            if(currentRoom.westExit != null) {
                System.out.print("west ");
            }
            System.out.println();
        }
    }

    // ... some code omitted ...
}
```

Both the `printWelcome` and `goRoom` methods contain the following lines of code:

```
System.out.println("You are " + currentRoom.getDescription());
System.out.print("Exits: ");
if(currentRoom.northExit != null) {
    System.out.print("north ");
}
if(currentRoom.eastExit != null) {
    System.out.print("east ");
}
if(currentRoom.southExit != null) {
    System.out.print("south ");
}
if(currentRoom.westExit != null) {
    System.out.print("west ");
}
System.out.println();
```

Code duplication is usually a symptom of bad cohesion. The problem here has its roots in the fact that both methods in question do two things: `printWelcome` prints the welcome message and prints the information about the current location, while `goRoom` changes the current location and then prints information about the (new) current location.

Both methods print information about the current location, but neither can call the other, because they also do other things. This is bad design.

A better design would use a separate, more cohesive method whose sole task is to print the current location information (Code 7.2). Both the `printWelcome` and `goRoom` methods can then make calls to this method when they need to print this information. This way, writing the code twice is avoided, and when we need to change it, we need to change it only once.

Code 7.2 `printLocationInfo` as a separate method

```
private void printLocationInfo()
{
    System.out.println("You are " +
        currentRoom.getDescription());
    System.out.print("Exits: ");
    if(currentRoom.northExit != null) {
        System.out.print("north ");
    }
    if(currentRoom.eastExit != null) {
        System.out.print("east ");
    }
```

Code 7.2 continued
`printLocationInfo` as a separate method

```
        if(currentRoom.southExit != null) {
            System.out.print("south ");
        }
        if(currentRoom.westExit != null) {
            System.out.print("west ");
        }
        System.out.println();
    }
```