Coupling between the `Game`, `Parser`, and `CommandWords` classes so far seems to have been very good – it was easy to make this extension, and we got it to work quickly.

The problem that was mentioned before – implicit coupling – becomes apparent when we now issue a *help* command. The output is

```
You are lost. You are alone. You wander
around at the university.

Your command words are:
  go quit help
```

Now we notice a small problem. The help text is incomplete: the new command, *look,* is not listed.

This seems easy to fix: we can just edit the help text string in the `Game`'s `printHelp` method. This is quickly done, and does not seem a great problem. But suppose we had not noticed this error now. Did you think of this problem before you just read about it here?

This is a fundamental problem, because every time a command is added the help text needs to be changed, and it is very easy to forget to make this change. The program compiles and runs, and everything seems fine. A maintenance programmer may well believe that the job is finished, and release a program that now contains a bug.

This is an example of implicit coupling. When commands change, the help text must be modified (coupling), but nothing in the program source clearly points out this dependence (thus implicit).

A well-designed class will avoid this form of coupling by following the rule of responsibility-driven design: since the `CommandWords` class is responsible for command words, it should also be responsible for printing command words. Thus we add the following method to the `CommandWords` class:

```
/**
 * Print all valid commands to System.out.
 */
public void showAll()
{
    for(String command : validCommands) {
        System.out.print(command + " ");
    }
    System.out.println();
}
```

The idea here is that the `printHelp` method in `Game`, instead of printing a fixed text with the command words, invokes a method that asks the `CommandWords` class to print all its command words. Doing this ensures that the correct command words will always be printed, and adding a new command will also add it to the help text without further change.

The only remaining problem is that the `Game` object does not have a reference to the `CommandWords` object. You can see in the class diagram (Figure 7.1) that there is no arrow from `Game` to `CommandWords`. This indicates that the `Game` class does not even know of the existence of the `CommandWords` class. Instead, the game just has a parser, and the parser has command words.

We could now add a method to the parser that hands the `CommandWords` object to the `Game` object, so that they could communicate. This would, however, increase the degree of coupling in our application: `Game` would then depend on `CommandWords`, which it currently does not. We would see this effect in the class diagram: `Game` would then have an arrow to `CommandWords`.

The arrows in the diagram are, in fact, a good first indication of how tightly coupled a program is: the more arrows, the more coupling. As an approximation of good class design, we can aim at creating diagrams with few arrows.

Thus the fact that `Game` did not have a reference to `CommandWords` is a good thing! We should not change this. From `Game`'s viewpoint, the fact that the `CommandWords` class exists is an implementation detail of the parser. The parser returns commands, and whether it uses a `CommandWords` object to achieve this or something else is entirely up to the parser's implementation.

It follows that a better design just lets the `Game` talk to the `Parser`, which in turn may talk to `CommandWords`. We can implement this by adding the following code to the `printHelp` method in `Game`:

```
System.out.println("Your command words are:");
parser.showCommands();
```

All that is missing then is the `showCommands` method in the `Parser`, which delegates this task to the `CommandWords` class. Here is the complete method (in class `Parser`):

```
/**
 * Print out a list of valid command words.
 */
public void showCommands()
{
    commands.showAll();
}
```