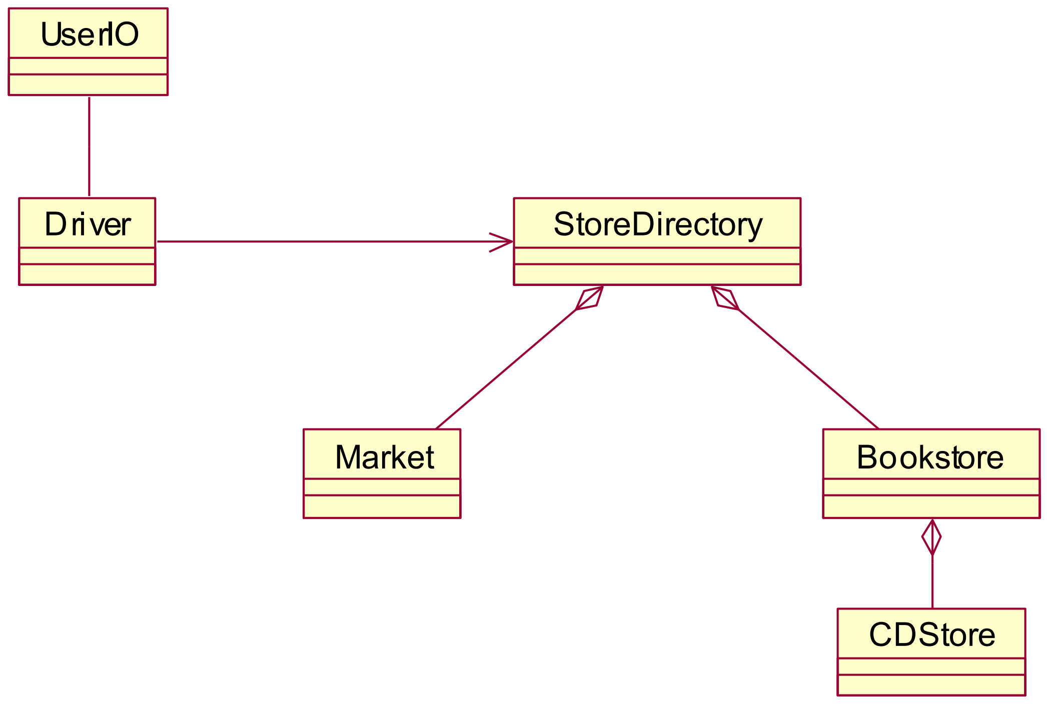
(Optional) Programming Exercise 3-5

This exercise is a miniature version of an online store. A user will have the option to get information about a food market, a bookstore and a CD store. The code (and instructions) you are given initially is designed to help you organize your code according to a well-known design pattern – the *façade pattern.* Using the pattern, a client class (in this case, a class that handles user requests) is able to gain access to a system of classes and services (in this case, info about various stores) only through a "façade" – a special class that provides controlled access to the services available from the collection of classes in the system. Here, the façade is the StoreDirectory class.

In this exercise, use the code in the package called `scope`. Here is a class diagram showing the relationships between the classes in this package. (Here, the UserIO class represents the system console window; in an optional later version of this lab, it will represent an actual Graphical User Interface class.)



The code for most of these classes has already been provided for you. You should NOT modify the code in any of the classes EXCEPT **`StoreDirectory`** and **`Driver`**.

The main method for this cluster of classes lies in `Driver`. When `Driver` starts up, it asks the user to input a 3-digit user ID in the console. After doing this, the user sees a set of options. The user types one of the letters A – H or N (to exit) to obtain the requested result. The user may continue after seeing an output result by typing 'Y'.

The `Driver` class is responsible for fulfilling user requests, and to accomplish its aim, it wants to access information stored in the `Market, Bookstore` and `CDStore` classes. But `Driver` lives in a different package from these classes and so does not have access to them (in effect, `Driver` doesn't know that these classes exist). The `StoreDirectory` class has been designed to provide public access to the data stored in these other classes. Therefore, `Driver` must fulfill requests by invoking the public methods available in `StoreDirectory`.

This exercise asks you to understand the relationship between `StoreDirectory` and `Market, Bookstore`, and `CDStore`, on the one hand, and the relationship between `Driver` and `StoreDirectory` on the other hand. To achieve this, I have asked you to write the code that implements these two relationships.

To get you started, I have implemented most of the methods in `StoreDirectory` and one of the methods in `Driver`. I have indicated with comments which methods need to be implemented by you.

A crude form of access control has been introduced in this code. If the 3-digit ID that you enter at the beginning is on a "bad list", you will not be able to gain access to the data that you request, in some cases. This is intended to show you one of the ways that "getters" and "setters" can be used to control access to private data fields.

Here is what you need to do for this assignment:

In the `StoreDirectory` class, implement the methods

```
      getNumCDsInBookstore
      addNewCD
      marketCarriesFoodItem
```

The first two of these will require you to work with the inner class CDStore. The third will require you to work with the Market class.

In the `Driver` class, the following methods have been only partially implemented:

```
      void displayNumberOfEmployees()
      void addEmployee(String employeeId)
      void addBook(String bookId)
      void addCD(String cdId)
      void checkIfBookIsInStock(String bookId)
      void displayNumberOfCDsInBookstore()
      void checkWhetherFoodItemInMarket(String foodItem)
```

Values that are used in these partial implementations are "fake" – the real values are stored or computed by accessing the `StoreDirectory` class. To see how to access `StoreDirectory`, use the implementation of

```
      displayNumberOfBooks()
```

in `Driver` as a guideline.