Do It

Go to the book's web site and run the example program. See if you can "break" the security built into the program. Remember, you can only attempt to be as secure as possible, nothing is 100% secure. Were you able to send harmful information to the program that affected the program? If so, what did you do? What might be missing from the program that allowed this to happen? If not, what stopped your harmful data from corrupting the program? What changes might you make to this program that would require you to include a data tier? What inefficiencies still exist in the program? What can you do to fix them?

Dependency Injection

As you have seen, when programs are created, a developer goes through many iterations before the final application has been completed. Along the way, experience programmers will keep different versions of their programs. This allows the programmers to back up to a previous version, quickly, if the version they are working on has too many significant problems. Otherwise, they would have to attempt to strip out the "bad" code without doing harm to the good code. Applications have many files (HTML, JavaScript, CSS, PHP classes, and PHP libraries). Keeping track of which version works with which, or easily changing one part of the program to use a new version of another part, can become confusing. Especially when the file names and class names are coded in the code of the program itself.

Chapter 2 briefly discussed *dependency injection*. It allows the program (client) that will use a block of code (such as a class) to not know the actual implementation of the block of code it will be using. The client program does not know the actual class name.

You can use this idea to help the development process. The example you are about to look at is not for large-scale applications. However, it does give you a chance to look at the benefits of dependency injection. Large-scale applications should use a MVC (Model-View-Control) model or an established tier (or component) system that provides more efficient communication between components.

> *Security and performance—Many times there is a trade-off between being as secure as possible and having a program that has the best performance possible. The example that follows will make multiple system calls when it checks to see if a program exists and then uses* `require_once` *to load the program. The check for file existence allows the program to handle missing files rather than the program just crashing. In a later chapter, you will look at using* `try/catch` *blocks to allow the program to capture any problems without crashing the program. This would reduce the number of system calls (you would no longer need to check for the existence of the files), giving the program better performance.*

The dog application uses classes and methods contained in different files. This requires the code to also include `require_once` statements to pull the files into the program. The current design places the actual file name in the `require_once` statement. This does not allow you to change different versions of the files, unless you change the code. You will now remove these dependencies to provide more flexibility for future development of the application.

Before you get too bogged down with the code, Figure 4-5 shows the flow of the relationship with the programs.