Managing Cairo Projects with Packages, Crates and Modules

As you write large programs, organizing your code will become increasingly important. By grouping related functionality and separating code with distinct features, you'll clarify where to find code that implements a particular feature and where to go to change how a feature works.

The programs we've written so far have been in one module in one file. As a project grows, you should organize code by splitting it into multiple modules and then multiple files. As a package grows, you can extract parts into separate crates that become external dependencies. This chapter covers all these techniques.

We'll also discuss encapsulating implementation details, which lets you reuse code at a higher level: once you've implemented an operation, other code can call your code without having to know how the implementation works.

A related concept is scope: the nested context in which code is written has a set of names that are defined as "in scope". When reading, writing, and compiling code, programmers and compilers need to know whether a particular name at a particular spot refers to a variable, function, struct, enum, module, constant, or other item and what that item means. You can create scopes and change which names are in or out of scope. You can't have two items with the same name in the same scope.

Cairo has a number of features that allow you to manage your code's organization. These features, sometimes collectively referred to as the _module system_, include:

- **Packages:** A Scarb feature that lets you build, test, and share crates.
- **Crates:** A tree of modules that corresponds to a single compilation unit. It has a root directory, and a root module defined at the _lib.cairo_ file under this directory.
- **Modules** and **use:** Let you control the organization and scope of items.
- **Paths:** A way of naming an item, such as a struct, function, or module.

In this chapter, we'll cover all these features, discuss how they interact, and explain how to use them to manage scope. By the end, you should have a solid understanding of the module system and be able to work with scopes like a pro!