7.12 Refactoring

Concept:

Refactoring is the activity of restructuring an existing design to maintain a good class design when the application is modified or extended.

When designing applications we should attempt to think ahead, anticipate possible changes in the future, and create highly cohesive, loosely coupled classes and methods that make modifications easy. This is a noble goal, but of course we cannot always anticipate all future adaptations, and it is not feasible to prepare for all possible extensions we can think of.

This is why *refactoring* is important.

Refactoring is the activity of restructuring existing classes and methods to adapt them to changed functionality and requirements. Often, in the lifetime of an application, functionality is gradually added. One common effect is that, as a side-effect of this, methods and classes slowly grow in length.

It is tempting for a maintenance programmer to add some extra code to existing classes or methods. Doing this for some time, however, decreases the degree of cohesion. When more and more code is added to a method or a class, it is likely that at some stage it will represent more than one clearly defined task or entity.

Refactoring is the rethinking and redesigning of class and method structures. Most commonly the effect is that classes are split into two, or that methods are divided into two or more methods. Refactoring can also include the joining of classes or methods into one, but that case is less common.

7.12.1 Refactoring and testing

Before we provide an example of refactoring, we need to reflect on the fact that when we refactor a program we are usually proposing to make some potentially large changes to something that already works. When something is changed there is a likelihood that errors will be introduced. Therefore it is important to proceed cautiously, and prior to refactoring we should establish that a set of tests exists for the current version of the program. If tests do not exist, then the first stage should be to create some tests that will be suitable for conducting regression testing on the refactored version. Only when these tests exist should the refactoring start. Ideally, the refactoring should then follow in two steps:

- The first step is to refactor in order to provide the same functionality as that of the original version. In other words, we restructure the source code to improve its quality, not to change or increase its functionality. Once this stage is completed, the regression tests should be run to ensure that we have not introduced unintended errors.
- The second step is taken only once we have re-established the baseline functionality in the refactored version. Then we are in a safe position to enhance the program. Once that has been done, of course, testing will need to be conducted on the new version.

Making several changes at the same time (refactoring and adding new features) makes it harder to locate the source of problems when they occur.