Defensive Programming: Introduction

Defensive programming is carefully guarded programming that helps you to construct reliable software by designing each component to protect itself as much as possible: for example, by checking that undocumented assumptions remain valid [Goodliffe 2007]. The guidelines in this chapter address areas of the Java language that can help to constrain the effect of an error or help to recover from an error.

Java language mechanisms should be used to limit the scope, lifetime, and accessibility of program resources. Also, Java annotations can be used to document the program, aiding readability and maintenance. Java programmers should be aware of implicit behaviors and avoid unwarranted assumptions about how the system behaves.

A good overall principle for defensive programming is simplicity. A complicated system is difficult to understand, difficult to maintain, and difficult to get right in the first place. If a construct turns out to be complicated to implement, consider redesigning or refactoring it to reduce the complexity.

Finally, the program should be designed to be as robust as possible. Wherever possible, the program should help the Java runtime system by limiting the resources it uses and by releasing acquired resources when they are no longer needed. Again, this can often be achieved by limiting the lifetime and accessibility of objects and other programming constructs. Not all eventualities can be anticipated, so a strategy should be developed to provide a graceful exit of last resort.