11.4 When to Use Exception Handling

Exception handling is designed to process **synchronous errors**, which occur when a statement executes. Common examples we'll see throughout the book are *out-of-range array indices*, *arithmetic overflow* (i.e., a value outside the representable range of values), *division by zero*, *invalid method parameters* and *thread interruption* (as we'll see in Chapter 23). Exception handling is not designed to process problems associated with **asynchronous events** (e.g., disk I/O completions, network message arrivals, mouse clicks and keystrokes), which occur in parallel with, and *independent of*, the program's flow of control.

Software Engineering Observation 11.3

Incorporate your exception-handling and error-recovery strategy into your system from the inception of the design process—including these after a system has been implemented can be difficult.

Software Engineering Observation 11.4

Exception handling provides a single, uniform technique for documenting, detecting and recovering from errors. This helps programmers working on large projects understand each other's error-processing code.

Software Engineering Observation 11.5

A great variety of situations can generate exceptions—some exceptions are easier to recover from than others.

Software Engineering Observation 11.6

Sometimes you can prevent an exception by validating data first. For example, before you perform integer division, you can ensure that the denominator is not zero, which prevents the `ArithmeticException` *that occurs when you divide by zero.*