11.9 Declaring New Exception Types

Most Java programmers use *existing* classes from the Java API, third-party vendors and freely available class libraries (usually downloadable from the Internet) to build Java applications. The methods of those classes typically are declared to throw appropriate exceptions when problems occur. You write code that processes these existing exceptions to make your programs more robust.

If you build classes that other programmers will use, it's often appropriate to declare your own exception classes that are specific to the problems that can occur when another programmer uses your reusable classes.

A New Exception Type Must Extend an Existing One

A new exception class must extend an existing exception class to ensure that the class can be used with the exception-handling mechanism. An exception class is like any other class; however, a typical new exception class contains no members other than four constructors:

- one that takes no arguments and passes a default error message `String` to the superclass constructor
- one that receives a customized error message as a `String` and passes it to the superclass constructor
- one that receives a customized error message as a `String` and a `Throwable` (for chaining exceptions) and passes both to the superclass constructor
- one that receives a `Throwable` (for chaining exceptions) and passes it to the superclass constructor.

Good Programming Practice 11.2

Associating each type of serious execution-time malfunction with an appropriately named `Exception` *class improves program clarity.*

Software Engineering Observation 11.15

Most programmers will not need to declare their own exception classes. Before defining your own, study the existing ones in the Java API and try to choose one that already exists. If there is not an appropriate existing class, try to extend a related exception class. For example, if you're creating a new class to represent when a method attempts a division by zero,

you might extend class `ArithmeticException` *because division by zero occurs during arithmetic. If the existing classes are not appropriate superclasses for your new exception class, decide whether your new class should be a checked or an unchecked exception class. If clients should be* required *to handle the exception, the new exception class should be a checked exception (i.e., extend* `Exception` *but not* `RuntimeException`*). The client application should be able to reasonably recover from such an exception. If the client code should be able to ignore the exception (i.e., the exception is an unchecked one), the new exception class should extend* `RuntimeException`.

Good Programming Practice 11.3

By convention, all exception-class names should end with the word `Exception`.