**EXECEPTION HANDLING**

Several lines of information are displayed in response to this invalid input. This information is known as a **stack trace**, which includes the name of the exception (java.lang.ArithmeticException) in a descriptive message that indicates the problem that occurred and the method-call stack (i.e., the call chain) at the time it occurred. The stack trace includes the path of execution that led to the exception method by method. This helps you debug the program.

In Java System.out.println() will print to the standard out of the system you are using. On the other hand, System.err.println() will print to the standard error.

If you are using a simple Java console application, both outputs will be the same (the command line or console) but you can reconfigure the streams so that for example, System.out still prints to the console but System.err writes to a file.

Also, IDEs like Eclipse show System.err in red text and System.out in black text by default.

use method nextInt to read an int value. Method nextInt throws an inputMismatchException if the value read in is not an integer.

System.out and System.err are **streams**—sequences of bytes. While System.out (known as the **standard output stream**) displays a program’s output, System.err (known as the **standard error stream**) displays a program’s errors. Output from these streams can be redirected (i.e., sent to somewhere other than the command prompt, such as to a file). Using two different streams enables you to easily separate errormessages from other output. For instance, data output from System.err could be sent to a log file, while data output from System.out can be displayed on the screen.

A throw statement specifies an object to be thrown. The operand of a throw can be of any class derived from class Throwable. *When toString is invoked on any Throwable object, its resulting string includes the descriptive string that was supplied to the constructor, or simply the class name if no string was supplied.* An object can be thrown without containing information about the problem that occurred. In this case, simply knowing that an exception of a particular type occurred may provide sufficient information for the handler to process the problem correctly.

If an exception has not been caught when control enters a finally block and the finally block throws an exception that’s not caught in the finally block, the first exception will be lost and the exception from the finally block will be returned to the calling method. Avoid placing code that can throw an exception in a finally block. If such code is required, enclose the code in a try…catch within the finally block.

Exception handling is intended to remove error-processing code from the main line of a program’s code to improve program clarity. Do not place try…catch…finally around every statement that may throw an exception. This makes programs difficult to read. Rather, place one try block around a significant portion of your code, follow that try block with catch blocks that handle each possible exception and follow the catch blocks with a single finally block (if one is required).

When an exception is thrown but not caught in a particular scope, the method-call stack is “unwound,” and an attempt is made to catch the exception in the next outer try block. This process is called **stack unwinding**. Unwinding the method-call stack means that the method in which the exception was not caught *terminates*, all local variables in that method go out of scope and control returns to the statement that originally invoked that method. If a try block encloses that statement, an attempt is made to catch the exception. If a try block does not enclose that statement or if the exception is not caught, stack unwinding occurs again.

Recall that exceptions derive from class Throwable. Class Throwable offers a **printStack- Trace** method that outputs to the standard error stream the stack trace. Often, this is helpful in testing and debugging. Class Throwable also provides a **getStackTrace** method that retrieves the stack-trace information that might be printed by printStackTrace. Class Throwable’s **getMessage** method returns the descriptive string stored in an exception.

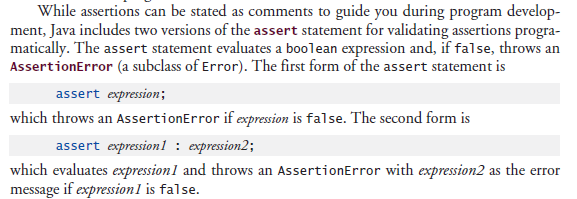
An exception that’s not caught in an application causes Java’s default exception handler to run. This displays the name of the exception, a descriptive message that indicates the problem that occurred and a complete execution stack trace. In an application with a single thread of execution, the application terminates. In an application with multiple threads, the thread that caused the exception terminates.

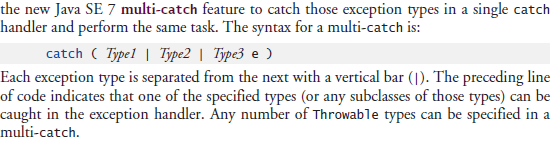
**Chained exceptions** enable an exception object to maintain the complete stack-trace information from the original exception.

A **precondition** must be true when a method is *invoked*. Preconditions describe constraints on method parameters and any other expectations the method has about the current state of a program just before it begins executing. If the preconditions are not met, then the method’s behavior is *undefined*—it may throw an exception, proceed with an illegal value or attempt to recover from the error. You should not expect consistent behavior if the preconditions are not satisfied.

A **postcondition** is true *after the method successfully returns*. Postconditions describe constraints on the return value and any other side effects the method may have. When defining a method, you should document all postconditions so that others know what to expect when they call your method, and you should make certain that your method honors all its postconditions if its preconditions are indeed met.

When implementing and debugging a class, it’s sometimes useful to state conditions that should be true at a particular point in a method. These conditions, called **assertions**, help ensure a program’s validity by catching potential bugs and identifying possible logic errors during development. Preconditions and postconditions are two types of assertions. Preconditions are assertions about its state when a method is invoked, and postconditions are assertions about a program’s state after a method finishes.

****

****

Typically resource-release code should be placed in a finally block to ensure that a resource is released, regardless of whether there were exceptions when the resource was used in the corresponding try block.

