11.13 Wrap-Up

In this chapter, you learned how to use exception handling to deal with errors. You learned that exception handling enables you to remove error-handling code from the “main line” of the program’s execution. We showed how to use `try` blocks to enclose code that may throw an exception, and how to use `catch` blocks to deal with exceptions that may arise.

You learned about the termination model of exception handling, which dictates that after an exception is handled, program control does not return to the throw point. We discussed checked vs. unchecked exceptions, and how to specify with the `throws` clause the exceptions that a method might throw.

You learned how to use the `finally` block to release resources whether or not an exception occurs. You also learned how to throw and rethrow exceptions. We showed how to obtain information about an exception using methods `printStackTrace`, `getStackTrace` and `getMessage`. Next, we presented chained exceptions, which allow you to wrap original exception information with new exception information. Then, we showed how to create your own exception classes.

We introduced preconditions and postconditions to help programmers using your methods understand conditions that must be true when the method is called and when it returns, respectively. When preconditions and postconditions are not met, methods typically throw exceptions. We discussed the `assert` statement and how it can be used to help you debug your programs. In particular, `assert` can be used to ensure that preconditions and postconditions are met.

We also introduced multi-`catch` for processing several types of exceptions in the same `catch` handler and the `try`-with-resources statement for automatically deallocating a resource after it's used in the `try` block. In the next chapter, we take a deeper look at graphical user interfaces (GUIs).