8. Error Handling

Lesson 2:
Exception fundamentals

8.4. Fundamentals of Exceptions
8.4.1. Catch or Specify Requirement

- Valid Java programming language code must honor the **Catch or Specify Requirement**. This means that code that might throw certain exceptions must be enclosed by either of the following:
  - A try statement that catches the exception. The try must provide a handler for the exception, as described in Catching and Handling Exceptions.
  - A method that specifies that it can throw the exception. The method must provide a throws clause that lists the exception, as described in Specifying the Exceptions Thrown by a Method.

- Code that fails to honor the Catch or Specify Requirement will not compile.

- Not all exceptions are subject to the Catch or Specify Requirement. To understand why, we need to look at the three basic **kinds of exceptions**, only one of which is subject to the Requirement.

8.4. Fundamentals of Exceptions
8.4.2. Kinds of Exceptions

- Java has three kinds of Exceptions: Checked Exceptions, Errors and Runtime Exceptions

- Checked Exceptions
  - These are exceptional conditions that a well-written application should anticipate and recover from.
  - For example, suppose an application prompts a user for an input file name, then opens the file by passing the name to the constructor for java.io.FileReader. Normally, the user provides the name of an existing, readable file, so the construction of the FileReader object succeeds, and the execution of the application proceeds normally. But sometimes the user supplies the name of a nonexistent file, and the constructor throws java.io.FileNotFoundException.

- A well-written program will catch this exception and notify the user of the mistake, possibly prompting for a corrected file name. Checked exceptions *are subject* to the Catch or Specify Requirement.

- All exceptions (except for those indicated by Error, RuntimeException, and their subclasses) are checked exceptions,.

8.4. Fundamentals of Exceptions
8.4.2. Kinds of Exceptions (cont…)

- Errors
  - These are exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from.
  - For example, suppose that an application successfully opens a file for input, but is unable to read the file because of a hardware or system malfunction. The unsuccessful read will throw java.io.IOError. An application might choose to catch this exception, in order to notify the user of the problem — but it also might make sense for the program to print a stack trace and exit.
- Errors *are not subject* to the Catch or Specify Requirement. Errors are those exceptions indicated by Error and its subclasses.
- Runtime Exceptions
  - These are exceptional conditions that are internal to the application, and that the application usually cannot anticipate or recover from. These usually indicate programming bugs, such as logic errors or improper use of an API.
  - For example, consider the application described previously that passes a file name to the constructor for FileReader. If a logic error causes a null to be passed to the constructor, the constructor will throw NullPointerException. The application can catch this exception, but it probably makes more sense to eliminate the bug that caused the exception to occur.