**Chapter 11**

**Exercises**

**11.14 (Exceptional Conditions)**

List the various exceptional conditions that have occurred in programs throughout this text so far. List as many additional exceptional conditions as you can. For each of these, describe briefly how a program typically would handle the exception by using the exception-handling techniques discussed in this chapter. Typical exceptions include division by zero and array index out of bounds.

**1. Division by Zero (ArithmeticException)**

* **Cause:** This happens when a number is divided by zero, which is mathematically undefined.
* **Handling:** A program can check if the denominator is zero before performing the division. If division by zero is detected, the program can display an error message and ask for a valid input.

**2. Array Index Out of Bounds (ArrayIndexOutOfBoundsException)**

* **Cause:** Trying to access an element in an array using an index that does not exist (e.g., accessing the 10th element in a 5-element array).
* **Handling:** The program should check the array's size before accessing elements. Using loops that respect the array’s boundaries can also prevent this error.

**3. Null Reference (NullPointerException)**

* **Cause:** When an object is null and the program tries to call a method or access a property on it.
* **Handling:** Before using an object, check if it is null. Ensuring that objects are properly initialized before use also prevents this error.

**4. Invalid Input (InputMismatchException)**

* **Cause:** When the user inputs data that does not match the expected type (e.g., entering text when a number is expected).
* **Handling:** The program should validate user input before using it. If the input is incorrect, prompt the user to enter the correct format.

**5. File Not Found (FileNotFoundException)**

* **Cause:** When a program tries to open a file that does not exist.
* **Handling:** Before opening a file, check if it exists. If it does not, display a meaningful message to the user and possibly allow them to specify a correct file path.

**6. Out of Memory (OutOfMemoryError)**

* **Cause:** When a program tries to allocate more memory than is available, often due to excessive object creation or infinite loops.
* **Handling:** Optimize memory usage by limiting large data structures, freeing unused objects, and avoiding infinite recursion or excessive loops.

**7. Class Not Found (ClassNotFoundException)**

* **Cause:** When a program tries to load a class dynamically that does not exist.
* **Handling:** Ensure that the class name is correct and that necessary dependencies are available in the program’s classpath.

**8. Illegal Argument (IllegalArgumentException)**

* **Cause:** When a method is given an invalid argument, such as a negative age or an unsupported value.
* **Handling:** Before passing arguments to methods, validate them to ensure they fall within the expected range.

**9. Number Format Exception (NumberFormatException)**

* **Cause:** Trying to convert a non-numeric string into a number, such as attempting to parse "abc" as an integer.
* **Handling:** Check if the input is numeric before attempting to convert it. If conversion fails, prompt the user for a valid number.

**10. Stack Overflow (StackOverflowError)**

* **Cause:** Infinite recursion or excessive method calls that exceed the call stack limit.
* **Handling:** Ensure that recursive functions have a proper base case and that loops terminate correctly.

**11. Interrupted Exception (InterruptedException)**

* **Cause:** When a thread is paused (e.g., sleeping) and another thread interrupts it.
* **Handling:** Properly handle thread interruptions by checking if a thread is supposed to stop before continuing execution.

**11.15 (Exceptions and Constructor Failure)**

Until this chapter, we’ve found dealing with errors detected by constructors to be a bit awkward. Explain why exception handling is an effective means for dealing with constructor failure.

Constructors are used to initialize objects when they are created. However, if something goes wrong during initialization—such as invalid input, insufficient resources, or file access issues—the object may not be properly created. Exception handling provides a structured way to manage such failures, ensuring program stability and preventing incomplete or corrupt objects from being used.

**1. Prevents Creation of Invalid Objects**

* If a constructor fails due to invalid parameters (e.g., setting a negative age in a Person class), an exception can be thrown instead of creating a faulty object.
* This prevents the program from working with an inconsistent or unusable object.

**Without exception handling:**

* The program might create a broken object that leads to undefined behavior later.

**With exception handling:**

* The program can detect the failure immediately and take corrective action.

**2. Ensures Resource Management**

* Some objects rely on external resources (files, databases, network connections). If a constructor fails while allocating such a resource, the failure must be handled properly.
* Exception handling ensures that resources are properly released if an error occurs.

For example:

* If a constructor tries to open a file that does not exist, an exception can be thrown instead of leaving an object in a half-initialized state.

**3. Avoids Partial Object Creation**

* Without exception handling, a constructor that fails halfway through its execution might leave the object in an inconsistent state.
* With exceptions, the failed object is never created, avoiding potential errors.

**4. Allows Constructor Delegation and Safe Recovery**

* If an object relies on another object in its constructor (e.g., a Car object needing an Engine object), and the dependency fails, exception handling allows graceful failure and error recovery.
* The program can catch the exception and either retry, provide a default value, or log the error for debugging.

**5. Enables Error Reporting and Debugging**

* Exception handling allows developers to log specific error messages and stack traces when a constructor fails.
* Instead of a silent failure, the program can provide meaningful feedback to help fix the issue.

**Conclusion**

Exception handling ensures that constructors either successfully initialize objects or signal a failure in a controlled manner. This prevents partially created objects, allows for proper resource management, and provides clear error reporting—making it an essential technique for handling constructor failures effectively.

**11.16 (Catching Exceptions with Superclasses)**

Use inheritance to create an exception superclass (called ExceptionA) and exception subclasses ExceptionB and ExceptionC, where ExceptionB inherits from ExceptionA and ExceptionC inherits from ExceptionB. Write a program to demonstrate that the catch block for type ExceptionA catches exceptions of types ExceptionB and ExceptionC.

**11.17 (Catching Exceptions Using Class Exception)**

Write a program that demonstrates how various exceptions are caught with catch (Exception exception) This time, define classes ExceptionA (which inherits from class Exception) and ExceptionB (which inherits from class ExceptionA). In your program, create try blocks that throw exceptions of types ExceptionA, ExceptionB, NullPointerException and IOException. All exceptions should be caught with catch blocks specifying type Exception.

**11.18 (Order of catch Blocks)**

Write a program demonstrating that the order of catch blocks is important. If you try to catch a superclass exception type before a subclass type, the compiler should generate errors.

**11.19 (Constructor Failure)**

Write a program that shows a constructor passing information about constructor failure to an exception handler. Define class SomeClass, which throws an Exception in the constructor. Your program should try to create an object of type SomeClass and catch the exception that’s thrown from the constructor.

**11.20 (Rethrowing Exceptions)**

Write a program that illustrates rethrowing an exception. Define methods someMethod and someMethod2. Method someMethod2 should initially throw an exception. Method someMethod should call someMethod2, catch the exception and rethrow it. Call someMethod from method main, and catch the rethrown exception. Print the stack trace of this exception.

**11.21 (Catching Exceptions Using Outer Scopes)**

Write a program showing that a method with its own try block does not have to catch every possible error generated within the try. Some exceptions can slip through to, and be handled in, other scopes.