23. Exception Handling

Java

Fall 2009

Instructor: Dr. Masoud Yaghini

Outline

- Exception-Handling Overview
- Example: No Exception Handling
- Example: Exception Handling
- When to Use Exception Handling
- References

Exception-Handling Overview

Introduction

- **Exception**
  - an indication of a problem that occurs during a program's execution
- **Exception handling**
  - resolving exceptions that may occur so program can continue or terminate well
- Exception handling enables programmers to create programs that are more robust

Examples

- `ArrayIndexOutOfBoundsException`
  - an attempt is made to access an element past the end of an array
- `NullPointerException`
  - when a `null` reference is used where an object is expected
- `InputMismatchException`
  - occurs when `Scanner` method `nextInt` receives a string that does not represent a valid integer
- `ArithmeticException`
  - can arise from a number of different problems in arithmetic

Exception-Handling Overview

- Intermixing program logic with error-handling logic can make programs difficult to read, modify, maintain and debug

- Exception handling enables programmers to remove error-handling code from the “main line” of the program’s execution

Performance Tip

- If the potential problems occur infrequently, intermixing program and error-handling logic can degrade a program's performance,

- Because the program must perform (potentially frequent) tests to determine whether the task executed correctly and the next task can be performed.

Example: No Exception Handling

Example: No Exception Handling

- **Thrown exception**
  - an exception that has occurred
- **Stack trace**
  - the information about exception, includes:
    - **Name of the exception** (e.g. java.lang.ArithmeticException) in a descriptive message that indicates the problem
    - **Complete method-call stack**
- **Throw point**
  - initial point at which the exception occurs, top row of call chain
- Example:
  - DivideByZeroNoExceptionHandling.java

DivideByZeroNoExceptionHandling.java

- Output 1:

Please enter an integer numerator: 100

Please enter an integer denominator: 7

Result: 100 / 7 = 14

DivideByZeroNoExceptionHandling.java

- Output 2:

Please enter an integer numerator: 100

Please enter an integer denominator: 0

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at DivideByZeroNoExceptionHandling.quotient(DivideByZeroNoExceptionHandling.java:11)
    at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:23)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    at java.lang.reflect.Method.invoke(Method.java:597)
    at com.intellij.rt.execution.application.AppMain.main(AppMain.java:90)
```

DivideByZeroNoExceptionHandling.java

- Output 3:

Please enter an integer numerator: 100

Please enter an integer denominator: hello

```
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:840)
        at java.util.Scanner.next(Scanner.java:1461)
        at java.util.Scanner.nextInt(Scanner.java:2091)
        at java.util.Scanner.nextInt(Scanner.java:2050)
        at DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:21)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
        at java.lang.reflect.Method.invoke(Method.java:597)
        at com.intellij.rt.execution.application.AppMain.main(AppMain.java:90)
```

Example: Exception Handling

Example: Exception Handling

- With exception handling, the program catches and handles the exception
- Next example allows user to try again if invalid input is entered (zero for denominator, or non-integer input)
- Example:
  - DivideByZeroWithExceptionHandling.java

DivideByZeroWithExceptionHandling.java

- **Output 1:**

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

DivideByZeroWithExceptionHandling.java

- Output 2:

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0

Exception: java.lang.ArithmeticException: / by zero
Zero is an invalid denominator. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

DivideByZeroWithExceptionHandling.java

- Output 3:

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello

Exception: java.util.InputMismatchException
You must enter integers. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

Enclosing Code in a try Block

- **try** block
  - encloses code that might throw an exception and the code that should not execute if an exception occurs
  - Consists of keyword **try** followed by a block of code enclosed in braces
  - `try` statement – consists of `try` block and corresponding `catch`

Catching Exceptions

- `catch` block
  - catches (i.e., receives) and handles an exception, contains:
  - Begins with keyword `catch`
  - Exception parameter in parentheses – exception parameter identifies the exception type and enables `catch` block to interact with caught exception object
  - Block of code in curly braces that executes when exception of proper type occurs

Catching Exceptions

- Matching `catch` block
  - the type of the exception parameter matches the thrown exception type exactly
- Uncaught exception
  - an exception that occurs for which there are no matching `catch` blocks

Common Programming Errors

- It is a syntax error to place code between a `try` block and its corresponding `catch` blocks.
- Each `catch` block can have only a single parameter—specifying a comma-separated list of exception parameters is a syntax error.

Termination Model of Exception Handling

- When an exception occurs:
  - `try` block terminates immediately
  - Program control transfers to first matching `catch` block
- After exception is handled:
  - Termination model of exception handling – program control does not return to the throw point because the `try` block has expired; Flow of control proceeds to the first statement after the last `catch` block
  - Resumption model of exception handling – program control resumes just after throw point

Using the throws Clause

- `throws` clause – specifies the exceptions a method may throws
  - Appears after method's parameter list and before the method's body
  - Contains a comma-separated list of exceptions
  - Exceptions can be thrown by statements in method's body of by methods called in method's body
  - Exceptions can be of types listed in `throws` clause or subclasses

When to Use Exception Handling

When to Use Exception Handling

- Exception handling is designed to process the errors, which occur when a statement executes.
- Examples:
  - out-of-range array indices
  - arithmetic overflow (i.e., a value outside the representable range of values)
  - division by zero
  - invalid method parameters

When to Use Exception Handling

- Exception handling is not designed to process problems which occur in parallel with, and independent of, the program's flow of control.
- Examples:
  - disk I/O completions
  - network message arrivals
  - mouse clicks and keystrokes

References

References

- H. M. Deitel and P. J. Deitel, **Java™ How to Program**, Sixth Edition, Prentice Hall, 2005. (Chapter 13)

The End