28. Exception Handling

Java

Summer 2008

Instructor: Dr. Masoud Yaghini

Outline

- Exception-Handling Overview
- Example: Divide By Zero
- Example: Handling ArithmeticExceptions and InputMismatchExceptions
- 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 gracefully
- Exception handling enables programmers to create programs that are more robust and fault-tolerant

Examples

- `ArrayIndexOutOfBoundsException` – an attempt is made to access an element past the end of an array
- `ClassCastException` – an attempt is made to cast an object that does not have an *is-a* relationship with the type specified in the cast operator
- `NullPointerException` – when a `null` reference is used where an object is expected

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: Divide By Zero

Example: Divide By Zero

- **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
- `ArithmeticException` – can arise from a number of different problems in arithmetic
- **Throw point** – initial point at which the exception occurs, top row of call chain
- `InputMismatchException` – occurs when `Scanner` method `nextInt` receives a string that does not represent a valid integer

Exception Handling

```
1  package chapter13;
2
3  // DivideByZeroNoExceptionHandling.java
4  // An application that attempts to divide by zero.
5  import java.util.Scanner;
6
7  public class DivideByZeroNoExceptionHandling
8  {
9     // demonstrates throwing an exception when a divide-by-zero occurs
10    public static int quotient( int numerator, int denominator )
11    {
12       return numerator / denominator; // possible division by zero
13    } // end method quotient
14
15    public static void main( String args[] )
16    {
17       Scanner scanner = new Scanner( System.in ); // scanner for input
18
19       System.out.print( "Please enter an integer numerator: " );
20       int numerator = scanner.nextInt();
21       System.out.print( "Please enter an integer denominator: " );
22       int denominator = scanner.nextInt();
23
24       int result = quotient( numerator, denominator );
25
26       System.out.printf(
27             "\nResult: %d / %d = %d\n", numerator, denominator, result );
28    } // end main
29 } // end class DivideByZeroNoExceptionHandling
```

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 chapter13.DivideByZeroNoExceptionHandling.quotient(DivideByZeroNoExceptionHandling.java:12)
    at chapter13.DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:24)
    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 chapter13.DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:22)
        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: Handling ArithmeticExceptions and InputMismatchExceptions

Exception Handling

Example: Handling ArithmeticExceptions and InputMismatchExceptions

- 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)

Exception Handling

```
1  package chapter13;
2
3  // DivideByZeroWithExceptionHandling.java
4  // An exception-handling example that checks for divide-by-zero.
5  import java.util.InputMismatchException;
6  import java.util.Scanner;
7
8  public class DivideByZeroWithExceptionHandling
9  {
10    // demonstrates throwing an exception when a divide-by-zero occurs
11    public static int quotient( int numerator, int denominator )
12          throws ArithmeticException
13    {
14       return numerator / denominator; // possible division by zero
15    } // end method quotient
16
17    public static void main( String args[] )
18    {
19       Scanner scanner = new Scanner( System.in ); // scanner for input
20       boolean continueLoop = true; // determines if more input is needed
21
```

Exception Handling

```
22      do
23      {
24         try // read two numbers and calculate quotient
25         {
26            System.out.print( "Please enter an integer numerator: " );
27            int numerator = scanner.nextInt();
28            System.out.print( "Please enter an integer denominator: " );
29            int denominator = scanner.nextInt();
30
31            int result = quotient( numerator, denominator );
32            System.out.printf( "\nResult: %d / %d = %d\n", numerator,
33               denominator, result );
34            continueLoop = false; // input successful; end looping
35         } // end try
36
37         catch ( InputMismatchException inputMismatchException )
38         {
39            System.err.printf( "\nException: %s\n",
40               inputMismatchException );
41            scanner.nextLine(); // discard input so user can try again
42            System.out.println(
43               "You must enter integers. Please try again.\n" );
44         } // end catch
```

```
45
46          catch ( ArithmeticException arithmeticException )
47          {
48             System.err.printf( "\nException: %s\n", arithmeticException );
49             System.out.println(
50                "Zero is an invalid denominator. Please try again.\n" );
51          } // end catch
52
53       } while ( continueLoop ); // end do...while
54    } // end main
55 } // end class DivideByZeroWithExceptionHandling
56
```

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

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
- Matching `catch` block – the type of the exception parameter matches the thrown exception type exactly or is a superclass of it
- Uncaught exception – an exception that occurs for which there are no matching `catch` blocks
  - Cause program to terminate if program has only one thread; Otherwise only current thread is terminated and there may be adverse effects to the rest of the program

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
- `try` statement – consists of `try` block and corresponding `catch`

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 **synchronous 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 associated with **asynchronous events,**
- 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