11

Exception Handling and Debugging

Last time

- OOP and polymorphism practice
- game programming

Objectives

- error handling and exceptions
- `try`, `catch`, `finally`, `throws`, and `throw` keywords
- Java exception hierarchy
- stack trace

Outline

Introduction

- Exception (derived from exceptional situation)
  - an indication of a problem that occurs during execution
- Exception handling
  - resolving exceptions that may occur so the program can continue or terminate gracefully

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
- `ClassCastException` – an attempt is made to cast an object that does not have an *is-a* relationship with the type

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
- Improves clarity

Example: Divide By Zero

- Thrown exception – an exception that has occurred
- Stack trace
  - Exception name in a descriptive message that indicates problem
  - Complete method-call stack
- Throw point – initial point at which the exception occurs, top row of call chain

```java
// Fig. 13.1: DivideByZeroNoExceptionHandling.java
// An application that attempts to divide by zero.
import java.util.Scanner;

public class DivideByZeroNoExceptionHandling
{
   // demonstrates throwing an exceptio
   public static int quotient( int numer
   {
      return numerator / denominator; // possible division by zero
   } // end method quotient

   public static void main( String args[] )
   {
      Scanner scanner = new Scanner( System.in ); // scanner for input

      System.out.print( "Please enter an integer numerator: " );
      int numerator = scanner.nextInt();
      System.out.print( "Please enter an integer
      int denominator = scanner.nextInt();

      int result = quotient( numerator, denominator );
      System.out.printf(
         "\nResult: %d / %d = %d\n", numerator, denominator, result );
   } // end main
} // end class DivideByZeroNoExceptionHandling
```

Attempt to divide; denominator may be zero

Read input; exception occurs if input is not a valid integer

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

Result: 100 / 7 = 14
```

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

Result: 100 / 7 = 14
```

```
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:10)
        at
DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:22)
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at
DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:20)
```

`try-catch-finally` block

- `try` **block – encloses code that might throw an exception and the code that should not execute in such a case**
- `catch` **block – catches and handles an exception**
- `finally` **block – release resources in certain situations to avoid resource leaks**

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 curly braces

Catching Exceptions

- `catch` **block – catches and handles an exception:**
  - **Begins with keyword** `catch`
  - **Exception parameter in parentheses – exception parameter identifies the exception type**
  - **Block of code in curly braces that executes when exception of proper type occurs**

Termination Model of Exception Handling

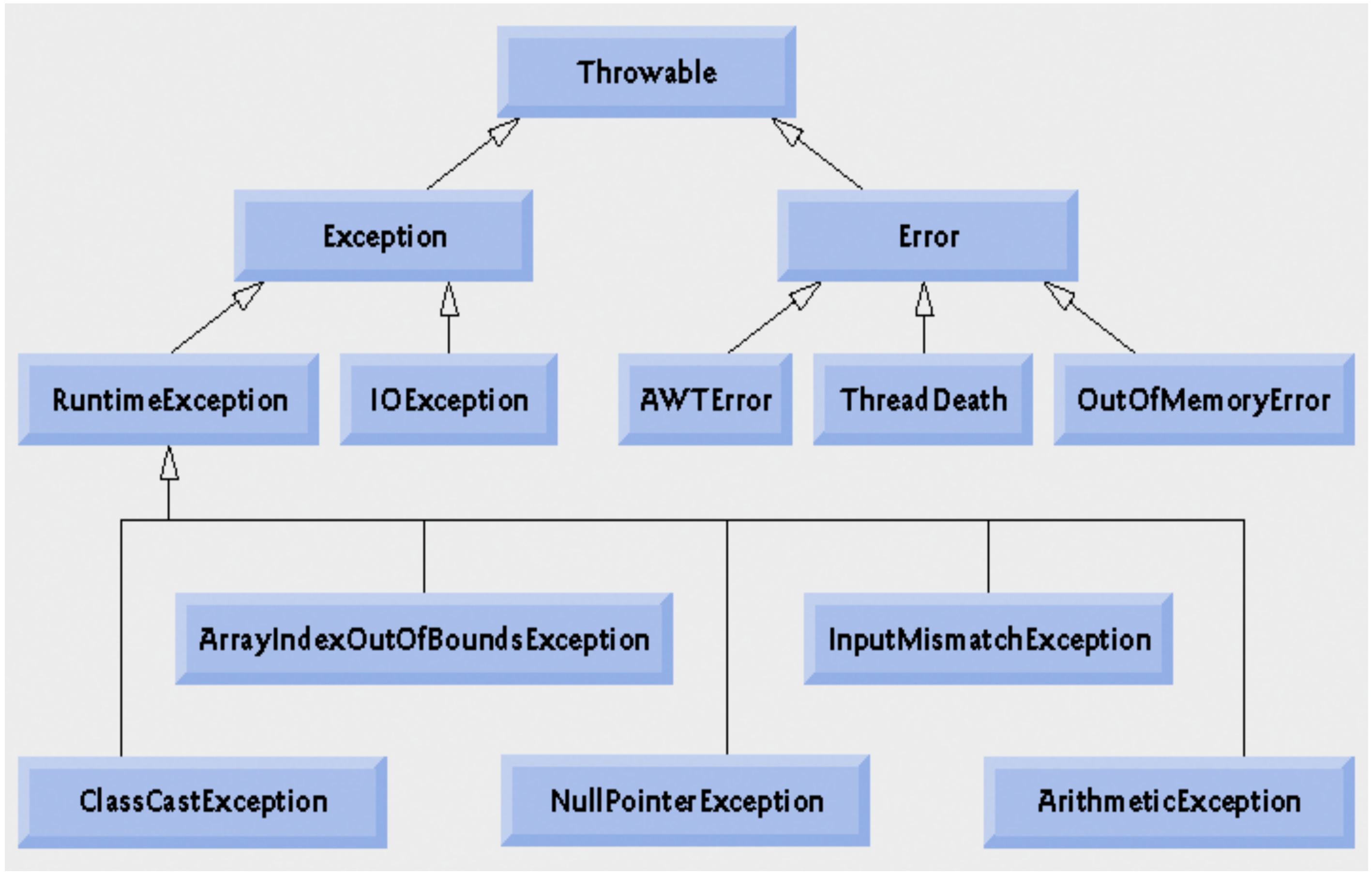
- When an exception occurs:
  - `try` block terminates immediately
  - Program control transfers to first matching `catch` block
- `try` statement – consists of `try` block and corresponding `catch` and/or `finally` blocks

`finally` block

- Programs that obtain certain resources must return them explicitly to avoid resource leaks using `finally` block
  - Consists of `finally` keyword followed by a block of code enclosed in curly braces
  - Optional in a `try` statement
  - If present, is placed after the last `catch` block
  - Executes whether or not an exception is thrown in the `try` block or any of its corresponding `catch` blocks

Java Exception Hierarchy

- Class `Throwable` is the superclass of all exceptions
  - Only `Throwable` objects can be used with the exception-handling mechanism
  - Has two subclasses: `Exception` and `Error` (JVM errors)
- Two categories of exceptions: checked and unchecked



Portion of class `Throwable`'s inheritance hierarchy.

Unchecked exceptions

- Inherit from class `RuntimeException` or class `Error`
- Compiler does not check code to see if exception is caught or declared
- If an unchecked exception occurs and is not caught, the program terminates
- Can typically be prevented by proper coding

Checked Exceptions

- Exceptions that inherit from class `Exception` but not from `RuntimeException`
- Compiler enforces a catch-or-declare requirement

Software Engineering Observation

Programmers are forced to deal with checked exceptions. This results in more robust code than would be created if programmers were able to simply ignore the exceptions.

Using the `throws` clause

- `throws` clause – specifies the exceptions that a method may throw
  - Appears after method's parameter list and before its 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

Using the `throw` statement

- `throw` statement – used to throw exceptions
- Programmers can thrown exceptions themselves from a method if something has gone wrong
- `throw` statement consists of keyword `throw` followed by the exception object

```
throw new Exception();
```

Declaring New Exception Types

- You can declare your own exception classes that are specific to your code (and classes)
- New exception class must extend an existing exception class
- Typically contains only two constructors
  - One takes no arguments, passes a default exception messages to the superclass constructor
  - One that receives a customized exception message as a string and passes it to the superclass constructor

Good Programming Practice

By convention, all exception-class names should end with the word `Exception`.