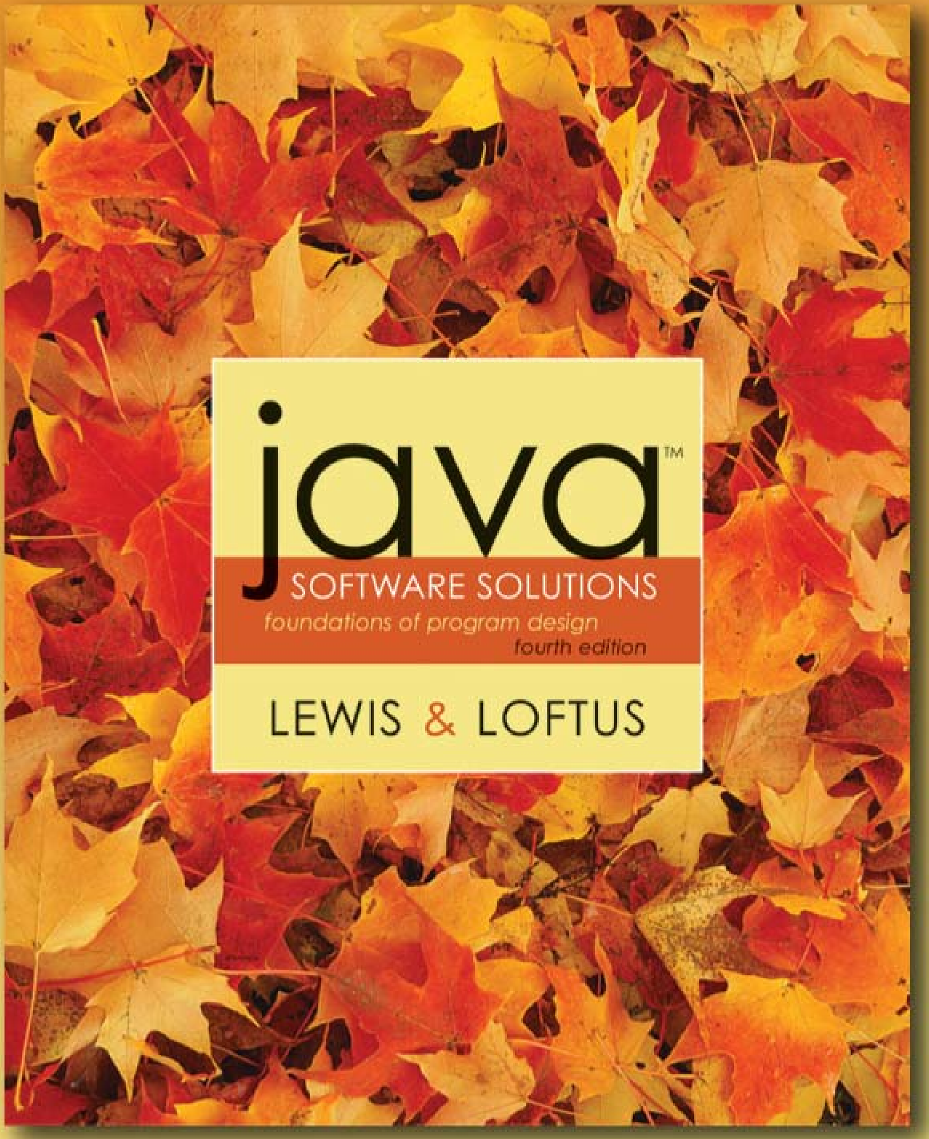
Lecture 10

Exceptions



© 2005 Pearson Addison-Wesley. All rights reserved.

Exceptions

- Exception handling is an important aspect of object-oriented design
- Lecture 10 focuses on:
  - the purpose of exceptions
  - exception messages
  - the try-catch statement
  - propagating exceptions

© 2004 Pearson Addison-Wesley. All rights reserved

Outline

Exception Handling

The try-catch Statement

Exception Classes

I/O Exceptions

© 2004 Pearson Addison-Wesley. All rights reserved

Exceptions

- An *exception* is an object that describes an unusual or erroneous situation
- Exceptions are *thrown* by a program, and may be *caught* and *handled* by another part of the program
- A program can be separated into a normal execution flow and an *exception execution flow*
- An *error* is also represented as an object in Java, but usually represents a unrecoverable situation and should not be caught

© 2004 Pearson Addison-Wesley. All rights reserved

Exception Handling

- Java has a predefined set of exceptions and errors that can occur during execution
- A program can deal with an exception in one of three ways:
  - ignore it
  - handle it where it occurs
  - handle it an another place in the program
- The manner in which an exception is processed is an important design consideration

© 2004 Pearson Addison-Wesley. All rights reserved

Exception Handling

- **If an exception is ignored by the program, the program will terminate abnormally and produce an appropriate message**
- **The message includes a *call stack trace* that:**
  - **indicates the line on which the exception occurred**
  - **shows the method call trail that lead to the attempted execution of the offending line**
- **See `Zero.java` (page 533)**

Outline

Exception Handling

➡ The try-catch Statement

Exception Classes

I/O Exceptions

© 2004 Pearson Addison-Wesley. All rights reserved

The try Statement

- To handle an exception in a program, the line that throws the exception is executed within a *try block*
- A try block is followed by one or more *catch* clauses
- Each catch clause has an associated exception type and is called an *exception handler*
- When an exception occurs, processing continues at the first catch clause that matches the exception type
- See `ProductCodes.java` (page 536)

The finally Clause

- A try statement can have an optional clause following the catch clauses, designated by the reserved word `finally`
- The statements in the finally clause always are executed
- If no exception is generated, the statements in the finally clause are executed after the statements in the try block complete
- If an exception is generated, the statements in the finally clause are executed after the statements in the appropriate catch clause complete

© 2004 Pearson Addison-Wesley. All rights reserved

Exception Propagation

- **An exception can be handled at a higher level if it is not appropriate to handle it where it occurs**

- **Exceptions *propagate* up through the method calling hierarchy until they are caught and handled or until they reach the level of the `main` method**

- **A try block that contains a call to a method in which an exception is thrown can be used to catch that exception**

- **See `Propagation.java` (page 539)**
- **See `ExceptionScope.java` (page 540)**

© 2004 Pearson Addison-Wesley. All rights reserved

Outline

© 2004 Pearson Addison-Wesley. All rights reserved

The Exception Class Hierarchy

- **Classes that define exceptions are related by inheritance, forming an exception class hierarchy**
- **All error and exception classes are descendents of the `Throwable` class**
- **A programmer can define an exception by extending the `Exception` class or one of its descendants**
- **The parent class used depends on how the new exception will be used**

© 2004 Pearson Addison-Wesley. All rights reserved

Checked Exceptions

- An exception is either *checked* or *unchecked*
- A *checked exception* either must be caught by a method, or must be listed in the *throws clause* of any method that may throw or propagate it
- A throws clause is appended to the method header
- The compiler will issue an error if a checked exception is not caught or asserted in a throws clause

© 2004 Pearson Addison-Wesley. All rights reserved

Unchecked Exceptions

- **An unchecked exception does not require explicit handling, though it could be processed that way**
- **The only unchecked exceptions in Java are objects of type `RuntimeException` or any of its descendants**
- **Errors are similar to `RuntimeException` and its descendants in that:**
  - **Errors should not be caught**
  - **Errors do not require a throws clause**

© 2004 Pearson Addison-Wesley. All rights reserved

The throw Statement

- Exceptions are thrown using the *throw* statement
- Usually a throw statement is executed inside an if statement that evaluates a condition to see if the exception should be thrown
- See `CreatingExceptions.java` (page 543)
- See `OutOfRangeException.java` (page 544)

© 2004 Pearson Addison-Wesley. All rights reserved

Outline

I/O Exceptions

- Let's examine issues related to exceptions and I/O
- A *stream* is a sequence of bytes that flow from a source to a destination
- In a program, we read information from an input stream and write information to an output stream
- A program can manage multiple streams simultaneously

© 2004 Pearson Addison-Wesley. All rights reserved

Standard I/O

- **There are three standard I/O streams:**
  - ***standard output* – defined by `System.out`**
  - ***standard input* – defined by `System.in`**
  - ***standard error* – defined by `System.err`**
- **We use `System.out` when we execute `println` statements**
- **`System.out` and `System.err` typically represent a particular window on the monitor screen**
- **`System.in` typically represents keyboard input, which we've used many times with Scanner objects**

The IOException Class

- **Operations performed by some I/O classes may throw an `IOException`**
  - **A file might not exist**
  - **Even if the file exists, a program may not be able to find it**
  - **The file might not contain the kind of data we expect**
- **An `IOException` is a checked exception**

© 2004 Pearson Addison-Wesley. All rights reserved

Writing Text Files

- In Chapter 5 we explored the use of the `Scanner` class to read input from a text file
- Let's now examine other classes that let us write data to a text file
- The `FileWriter` class represents a text output file, but with minimal support for manipulating data
- Therefore, we also rely on `PrintStream` objects, which have `print` and `println` methods defined for them

© 2004 Pearson Addison-Wesley. All rights reserved

Writing Text Files

- Finally, we'll also use the `PrintWriter` class for advanced internationalization and error checking
- We build the class that represents the output file by combining these classes appropriately
- See `TestData.java` (page 547)
- Output streams should be closed explicitly

Summary

- **Lecture 10 has focused on:**
  - **the purpose of exceptions**
  - **exception messages**
  - **the try-catch statement**
  - **propagating exceptions**
  - **the exception class hierarchy**

© 2004 Pearson Addison-Wesley. All rights reserved