Exceptions

Exceptions

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

Outline

➡ **Exception Handling**

The try-catch Statement

Exception Classes

I/O Exceptions

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**

Exception Handling

- The Java API has a predefined set of exceptions 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

Exception Handling

- If an exception is ignored (not caught) by the program, the program will terminate 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

```java
//********************************************************************
//  Zero.java       Author: Lewis/Loftus
//
//  Demonstrates an uncaught exception.
//********************************************************************

public class Zero
{
   //-----------------------------------------------------------------
   //  Deliberately divides by zero to produce an exception.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int numerator = 10;
      int denominator = 0;

      System.out.println (numerator / denominator);

      System.out.println ("This text will not be printed.");
   }
}
```

Output (when program terminates)

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Zero.main(Zero.java:17)
```

```java
public class Zero
{
   //-----------------------------------------------------------------
   //  Deliberately divides by zero to produce an exception.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int numerator = 10;
      int denominator = 0;

      System.out.println (numerator / denominator);

      System.out.println ("This text will not be printed.");
   }
}
```

Outline

The try Statement

- To handle an exception in a program, use a **try-catch statement**
- 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 within the try block, processing immediately jumps to the first catch clause that matches the exception type
- See ProductCodes.java

- 13 character product code
- Zone: 10th character
- District: 4 digit number (starting at the 4th character or the code)

```java
//********************************************************************
//  ProductCodes.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a try-catch block.
//********************************************************************

import java.util.Scanner;

public class ProductCodes
{
   //-----------------------------------------------------------------
   //  Counts the number of product codes that are entered with a
   //  zone of R and district greater than 2000.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String code;
      char zone;
      int district, valid = 0, banned = 0;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter product code (XXX to quit): ");
      code = scan.nextLine();
```

continue

continue

```
      while (!code.equals ("XXX"))
      {
         try
         {
            zone = code.charAt(9);
            district = Integer.parseInt(code.substring(3, 7));
            valid++;
            if (zone == 'R' && district > 2000)
               banned++;
         }
         catch (StringIndexOutOfBoundsException exception)
         {
            System.out.println ("Improper code length: " + code);
         }
         catch (NumberFormatException exception)
         {
            System.out.println ("District is not numeric: " + code);
         }

         System.out.print ("Enter product code (XXX to quit): ");
         code = scan.nextLine();
      }

      System.out.println ("# of valid codes entered: " + valid);
      System.out.println ("# of banned codes entered: " + banned);
   }
}
```

continue

```
      wh
      {
```

Sample Run

```
Enter product code (XXX to quit): TRV2475A5R-14
Enter product code (XXX to quit): TRD1704A7R-12
Enter product code (XXX to quit): TRL2k74A5R-11
District is not numeric: TRL2k74A5R-11
Enter product code (XXX to quit): TRQ2949A6M-04
Enter product code (XXX to quit): TRV2105A2
Improper code length: TRV2105A2
Enter product code (XXX to quit): TRQ2778A7R-19
Enter product code (XXX to quit): XXX
# of valid codes entered: 4
# of banned codes entered: 2
```

```
         }
         catch (NumberFormatException exception)
         {
            System.out.println ("District is not numeric: " + code);
         }

         System.out.print ("Enter product code (XXX to quit): ");
         code = scan.nextLine();
      }

      System.out.println ("# of valid codes entered: " + valid);
      System.out.println ("# of banned codes entered: " + banned);
   }
}
```

The finally Clause

- A try statement can have an optional **finally clause**, which is **always executed**
- If no exception is generated, the statements in the finally clause are executed after the statements in the try block finish
- **If an exception is generated, the statements in the finally clause are executed after the statements in the appropriate catch clause finish**

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
- See Propagation.java
- See ExceptionScope.java

```java
//********************************************************************
//  Propagation.java       Author: Lewis/Loftus
//
//  Demonstrates exception propagation.
//********************************************************************

public class Propagation
{
   //-----------------------------------------------------------------
   //  Invokes the level1 method to begin the exception demonstration.
   //-----------------------------------------------------------------
   static public void main (String[] args)
   {
      ExceptionScope demo = new ExceptionScope();

      System.out.println("Program beginning.");
      demo.level1();
      System.out.println("Program ending.");
   }
}
```

```java
//********************************************************************
//  ExceptionScope.java       Author: Lewis/Loftus
//
//  Demonstrates exception propagation.
//********************************************************************

public class ExceptionScope
{
   //-----------------------------------------------------------------
   //  Catches and handles the exception that is thrown in level3.
   //-----------------------------------------------------------------
   public void level1()
   {
      System.out.println("Level 1 beginning.");

      try
      {
         level2();
      }
      catch (ArithmeticException problem)
      {
         System.out.println ();
         System.out.println ("The exception message is: " +
                             problem.getMessage());
         System.out.println ();
```

continue

continue

```
         System.out.println ("The call stack trace:");
         problem.printStackTrace();
         System.out.println ();
      }

      System.out.println("Level 1 ending.");
   }

   //-----------------------------------------------------------------
   //  Serves as an intermediate level.  The exception propagates
   //  through this method back to level1.
   //-----------------------------------------------------------------
   public void level2()
   {
      System.out.println("Level 2 beginning.");
      level3 ();
      System.out.println("Level 2 ending.");
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Performs a calculation to produce an exception.  It is not
   //  caught and handled at this level.
   //-----------------------------------------------------------------
   public void level3 ()
   {
      int numerator = 10, denominator = 0;

      System.out.println("Level 3 beginning.");
      int result = numerator / denominator;
      System.out.println("Level 3 ending.");
   }
}
```

Output

```
Program beginning.
Level 1 beginning.
Level 2 beginning.
Level 3 beginning.

The exception message is: / by zero

The call stack trace:
java.lang.ArithmeticException: / by zero
        at ExceptionScope.level3(ExceptionScope.java:54)
        at ExceptionScope.level2(ExceptionScope.java:41)
        at ExceptionScope.level1(ExceptionScope.java:18)
        at Propagation.main(Propagation.java:17)

Level 1 ending.
Program ending.
```

Outline

Exception Handling

The try-catch Statement

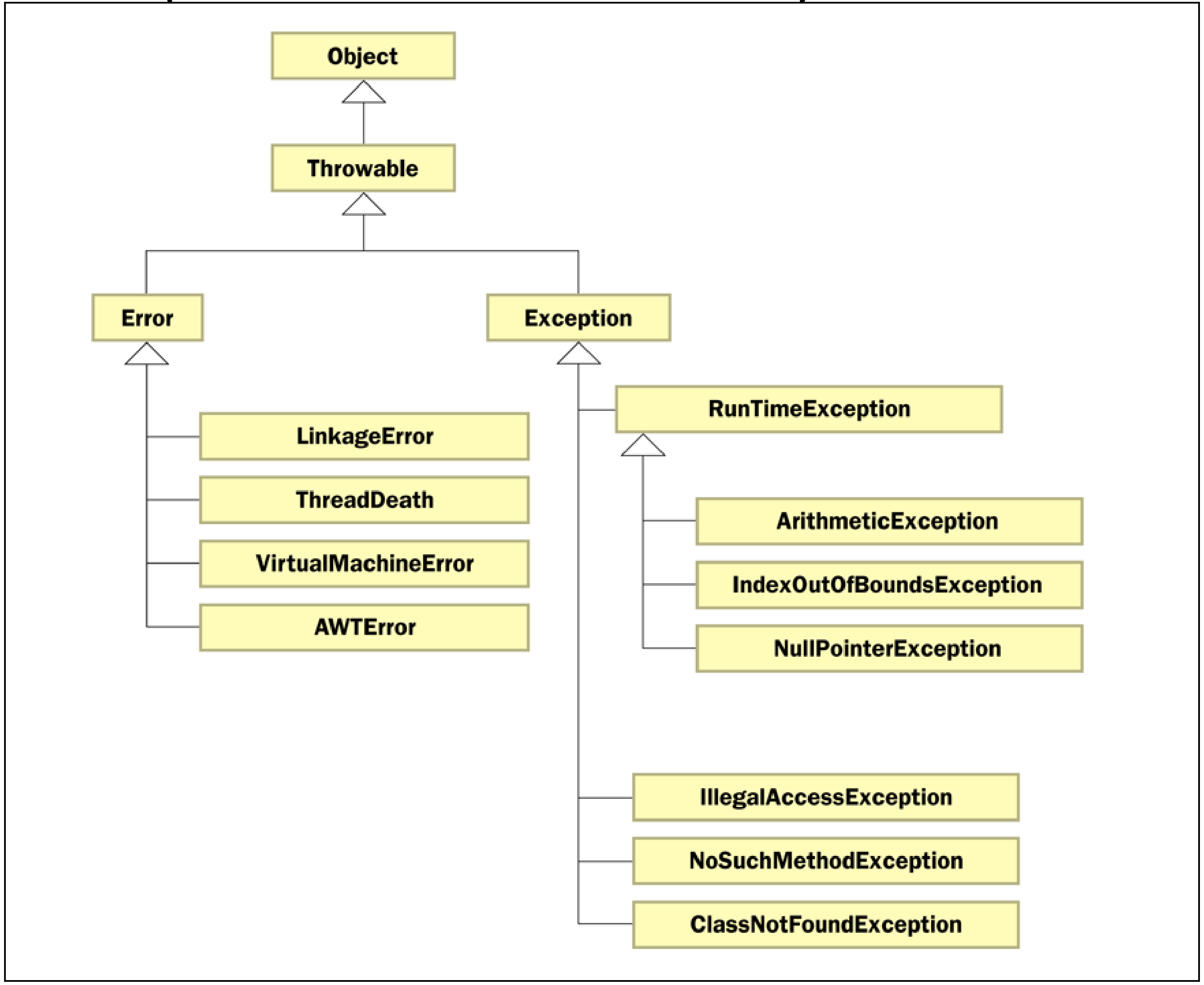
➡ Exception Classes

I/O Exceptions

The Exception Class Hierarchy

- Exception classes in the Java API 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

The Exception Class Hierarchy



Checked Exceptions

- **An exception is either checked or unchecked**
- A checked exception must either be caught 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 listed in a throws clause

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

Quick Check

Which of these exceptions are checked and which are unchecked?

`NullPointerException`

`IndexOutOfBoundsException`

`ClassNotFoundException`

`NoSuchMethodException`

`ArithmeticException`

Quick Check

Which of these exceptions are checked and which are unchecked?

`NullPointerException`	**`Unchecked`**
`IndexOutOfBoundsException`	**`Unchecked`**
`ClassNotFoundException`	**`Checked`**
`NoSuchMethodException`	**`Checked`**
`ArithmeticException`	**`Unchecked`**

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
- See OutOfRangeException.java

```java
//********************************************************************
//  CreatingExceptions.java       Author: Lewis/Loftus
//
//  Demonstrates the ability to define an exception via inheritance.
//********************************************************************

import java.util.Scanner;

public class CreatingExceptions
{
   //-----------------------------------------------------------------
   //  Creates an exception object and possibly throws it.
   //-----------------------------------------------------------------
   public static void main (String[] args) throws OutOfRangeException
   {
      final int MIN = 25, MAX = 40;

      Scanner scan = new Scanner (System.in);

      OutOfRangeException problem =
         new OutOfRangeException ("Input value is out of range.");
```

continue

continue

```
      System.out.print ("Enter an integer value between " + MIN +
                        " and " + MAX + ", inclusive: ");
      int value = scan.nextInt();

      //  Determine if the exception should be thrown
      if (value < MIN || value > MAX)
         throw problem;

      System.out.println ("End of main method.");  // may never reach
   }
}
```

Sample Run

```
Enter an integer value between 25 and 40, inclusive: 69
Exception in thread "main" OutOfRangeException:
     Input value is out of range.
     at CreatingExceptions.main(CreatingExceptions.java:20)
```

```
      if (value < MIN || value > MAX)
         throw problem;

      System.out.println ("End of main method.");  // may never reach
   }
}
```

```
//********************************************************************
//  OutOfRangeException.java       Author: Lewis/Loftus
//
//  Represents an exceptional condition in which a value is out of
//  some particular range.
//********************************************************************

public class OutOfRangeException extends Exception
{
   //-----------------------------------------------------------------
   //  Sets up the exception object with a particular message.
   //-----------------------------------------------------------------
   OutOfRangeException (String message)
   {
      super (message);
   }
}
```

Quick Check

What is the matter with this code?

```
System.out.println("Before throw");
throw new OutOfRangeException("Too High");
System.out.println("After throw");
```

Quick Check

What is the matter with this code?

```
System.out.println("Before throw");
throw new OutOfRangeException("Too High");
System.out.println("After throw");
```

The throw is not conditional and therefore always occurs. The second `println` statement can never be reached.

Outline

Exception Handling

The try-catch Statement

Exception Classes

➡ I/O Exceptions

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

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 the console window
- System.in typically represents keyboard input, which we've used many times with Scanner

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**

Writing Text Files

- 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

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
- Output streams should be closed explicitly
- See TestData.java

```
//********************************************************************
//  TestData.java       Author: Lewis/Loftus
//
//  Demonstrates I/O exceptions and the use of a character file
//  output stream.
//********************************************************************

import java.util.Random;
import java.io.*;

public class TestData
{
   //-----------------------------------------------------------------
   //  Creates a file of test data that consists of ten lines each
   //  containing ten integer values in the range 10 to 99.
   //-----------------------------------------------------------------
   public static void main (String[] args) throws IOException
   {
      final int MAX = 10;

      int value;
      String file = "test.dat";

      Random rand = new Random();
```

continue

continue

```
      FileWriter fw = new FileWriter (file);
      BufferedWriter bw = new BufferedWriter (fw);
      PrintWriter outFile = new PrintWriter (bw);

      for (int line=1; line <= MAX; line++)
      {
         for (int num=1; num <= MAX; num++)
         {
            value = rand.nextInt (90) + 10;
            outFile.print (value + "   ");
         }
         outFile.println ();
      }

      outFile.close();
      System.out.println ("Output file has been created: " + file);
   }
}
```

continue

```
      FileW
      BufferedWriter bw = new BufferedWriter (fw);
      PrintWriter outFile = new PrintWriter (bw);

      for (int line=1; line <= MAX; line++)
      {



      }

      ou
      Sy                                                        );
   }
}
```

Output

```
Output file has been created: test.dat
```

Sample test.dat File

```
77    46    24    67    45    37    32    40    39    10
90    91    71    64    82    80    68    18    83    89
25    80    45    75    74    40    15    90    79    59
44    43    95    85    93    61    15    20    52    86
60    85    18    73    56    41    35    67    21    42
93    25    89    47    13    27    51    94    76    13
33    25    48    42    27    24    88    18    32    17
71    10    90    88    60    19    89    54    21    92
45    26    47    68    55    98    34    38    98    38
48    59    90    12    86    36    11    65    41    62
```

Exception Example

```
public class DivideException {

   public static void main(String[] args) {
          division(100,4);          // Line 1
          division(100,0);          // Line 2
      System.out.println("Exit main().");
   }

   public static void division(int totalSum, int totalNumber) {
          System.out.println("Computing Division.");
          int average  = totalSum/totalNumber;
      System.out.println("Average : "+ average);
   }
}
```

Output

```
Computing Division.
Average : 25
Computing Division.
java.lang.ArithmeticException: / by zero
at DivideException.division(DivideException.java:11)
at DivideException.main(DivideException.java:5)
Exception in thread "main"
```

```
public class DivideException2 {
   public static void main(String[] args) {
              int result  = division(100,0);        // Line 2
      System.out.println("result : "+result);
   }

   public static int division(int totalSum, int totalNumber) {
              int quotient = -1;
              System.out.println("Computing Division.");
              try{
                            quotient  = totalSum/totalNumber;

              }
              catch(Exception e){
                            System.out.println("Exception : "+ e.getMessage());
              }
              finally{
                            if(quotient != -1){
                                          System.out.println("Finally Block Executes");
                                          System.out.println("Result : "+ quotient);
                            }else{
                                          System.out.println("Finally Block Executes. Exception Occurred");
                                          return quotient;
                            }
              }
              return quotient;
   }
}
```

Output

```
Computing Division.
Exception : / by zero
Finally Block Executes. Exception Occurred
result : -1
```

Exception Example

```
public class Prop
{
static public void main (String[] args)
  {
    Test demo = new Test();
    System.out.println(“1");
    demo.a();
    System.out.println(“2");
  }
}

public class Test
{
  public void a()
  {
    System.out.println(“3");
    try{
              b();
             } catch (ArithmeticException problem)
    {
              System.out.println ("The exception message is: " +
problem.getMessage());
             }
    System.out.println(“4");
  }
```

```
// Test class continues here…..
public void b()
  {
    System.out.println(“5");
    c();
    System.out.println(“6");
  }
public void c()
  {
    int numerator = 10, denominator = 0;
    System.out.println(“7");
    int result = numerator / denominator;
    System.out.println(“8");
  }
}
```

Answer

```
1
3
5
7
The exception message is: / by zero
4
2
```