9.3 Using Exception Classes
Passing the Buck—Declaring Exceptions

When defining a method, you must include a `throws`-clause to declare any exception that might be thrown but is not caught in the method.

- Use a *`throws`-clause* to "**pass the buck**" to whatever method calls it (**pass the** [illegible] for the `catch` block to the method that **calls it**)
  - » that method can also pass the buck,
    but eventually some method must catch it

- This tells other methods
  "If you call me, you **must** [illegible] that I throw."

Example: **throws**-Clause

`DoDivision`

- It may throw a `DivideByZeroException` in the method `normal`
- But the `catch` block is in `main`
- So `normal` must include in the first line of the method definition:

```
public void normal() throws DivideByZeroException
{
    <statements to define the normal method>
}
```

Listing 9.7 Passing the Buck with a throws Clause - DoDivision.java

```
import java.util.Scanner;

public class DoDivision
{
   private int numerator;
   private int denominator;
   private double quotient;

   public static void main(String[] args)
   {
       DoDivision doIt = new DoDivision( );
```

```
    try
    {
        doIt.doNormalCase( );
    }
    catch(DivideByZeroException e)
    {
        System.out.println(e.getMessage( ));
        doIt.giveSecondChance( );
    }

    System.out.println("End of Program.");
}

public void doNormalCase( ) throws DivideByZeroException
{
    System.out.println("Enter numerator:");
    Scanner keyboard = new Scanner(System.in);
    numerator = keyboard.nextInt( );

    System.out.println("Enter denominator:");
    denominator = keyboard.nextInt( );
    if (denominator == 0)
        throw new DivideByZeroException( );
    quotient = numerator / (double)denominator;
                System.out.println(numerator + "/" + denominator +
                                                    " = " + quotient);
}
```

```
public void giveSecondChance( )
 {
     System.out.println("Try Again:");
     System.out.println("Enter numerator:");
     Scanner keyboard = new Scanner(System.in);

     numerator = keyboard.nextInt( );
     System.out.println("Enter denominator:");
     System.out.println("Be sure the denominator is not zero.");
     denominator = keyboard.nextInt( );

     if (denominator == 0)
     {
        System.out.println("I cannot do division by zero.");
        System.out.println("Since I cannot do what you want,");
        System.out.println("the program will now end.");
        System.exit(0);
     }

     quotient = ((double)numerator) / denominator;
                 System.out.println(numerator + "/" + denominator +
                                                  " = " + quotient);
  }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
Enter numerator:
5
Enter denominator:
0
Dividing by Zero!
Try Again:
Enter numerator:
5
Enter denominator:
Be sure the denominator is not zero.
0
I cannot do division by zero.
Since I cannot do what you want,
the program will now end.
계속하려면 아무 키나 누르십시오 . . .
```

Example

```
public static void main(String[] args)
 {
     DoDivision2 doIt = new DoDivision2( );

     doIt.normal( );

     System.out.println("End of Program.");

  }

  public void normal( ) throws DivideByZeroException
  {
    System.out.println("Enter numerator:");
     Scanner keyboard = new Scanner(System.in);
     numerator = keyboard.nextInt( );

     System.out.println("Enter denominator:");
      ......................


   }
```

```
D:\My Documents\@@@@@@jv\ch08\DoDivision2.java:13: unreported
exception DivideByZeroException; [redacted] to be
thrown
        doIt.normal( );
            ^
1 error

Tool completed with exit code 1
```

More about Passing the Buck

Good programming practice:

Every exception thrown should eventually be caught in some method

Most Exceptions that might be thrown when a method is invoked must be accounted for in one of two ways.

- 1) Normally exceptions are either caught
  - » The Possible exception can be caught in a catch block within the method definition
- 2) or *deferred* to the calling method in
  - » letting whoever uses the method worry about how to handle the exception.
- If a method throws an exception, it expects the catch block to be in that method unless it is deferred by a `throws`-clause
  - » if the calling method also defers with a `throws`-clause, its calling program is expected to have the `catch` block, etc., up the line all the way to `main`, until a `catch` block is found

Multiple Exceptions and `catch` Blocks in a Method

- Methods can throw more than one exception
- The `catch` blocks immediately following the try block are searched in sequence for one that catches the exception type
  - » the first catch block that handles the exception type is the only one that executes
- Specific exceptions are derived from more general types
  - » both the specific and general types from which they are derived will handle exceptions of the more specific type
- So put the `catch` blocks for the more specific, derived, exceptions early and the more general ones

Catch the more exception first.

Listing 9.8 Catching Multiple Exceptions - TwoCatchesDemo.java

```
 // Listing 9.8 Catching Multiple Exceptions
// THis is just a sample example for learning the basic
// syntax for exception handlings

public class TwoCatchesDemo
{
   public static void main(String[] args)
   {
      try
      {
         int widgets, defective;
         double ratio;

         System.out.println("Enter number of widgets produced:");
         widgets = SavitchIn.readLineInt( );
```

```
if (widgets < 0)
        throw new NegativeNumberException("widgets");
      System.out.println("How many were defective?");

       Scanner keyboard = new Scanner(System.in);
      int widgets = keyboard.nextInt( );

      if (defective < 0)
        throw new NegativeNumberException("defective widgets");

      ratio = exceptionalDivision(widgets, defective);
      System.out.println( "One in every  " + ratio
                          + " widgets is defective.");
    }
    catch(DivideByZeroException e)
    {
      System.out.println("Congratulations! A perfect record!");
    }
    catch(NegativeNumberException e)
    {
      System.out.println("Cannot have a negative number of "
                         + e.getMessage( ));
    }

    System.out.println("End of program.");
  }
```

```
public static double exceptionalDivision(double numerator,
                double denominator) throws DivideByZeroException
  {
      if (denominator == 0)
          throw new DivideByZeroException( );
     return (numerator/denominator);
  }
}
```

C:₩WINDOWS₩system32₩cmd.exe

```
Enter number of widgets produced:
1000
How many were defective?
0
Congratulations! A perfect record!
End of program.
계속하려면 아무 키나 누르십시오 . . .
```

Listing 9.9 The Class NegativeNumberException - NegativeNumberException.java

```
// Listing 9.9  The Class NegativeNumberException

public class NegativeNumberException extends Exception
{
   public NegativeNumberException( )
   {
      super("Negative Number Exception!");
   }
   public NegativeNumberException(String message)
   {
      super(message);
   }
}
```

Catch the More Exception First

```
Catch (Exception e)
{

…

}

Catch (DivideByZeroException e)
{


..

}
```

```
Catch (DivideByZeroException e)
{


..

}


Catch (Exception e)
{

…

}
```

The `finally` Block

At this stage of your programming you may not have much use for the `finally` block, but it is included for _____ - you may have find it useful later

- You can add a `finally` block after the `try`/`catch` blocks
- `finally` blocks execute _____ `catch` block(s) execute
- Code organization using `finally` block:

```
try block
catch block
finally
{
   <Code to be executed whether or not an exception is thrown>
}
```

Three Possibilities for a **`try-catch-finally`** Block

- 1) The `try`-block runs to the end and no exception is thrown.
  - » The `finally`-block runs after the `try`-block.
- 2) An exception is thrown in the `try`-block and caught in the matching `catch`-block.
  - » The `finally`-block runs after the `catch`-block.
- 3) An exception is thrown in the `try`-block and there is no matching `catch`-block.
  - » The `finally`-block is executed before the method ends.
  - » Code that is after the `catch`-blocks but not in a `finally`-block would not be executed in this situation.

Case Study: A Line-Oriented Calculator

- Preliminary version with no exception handling written first
- Exception when user enters unknown operator
- Three choices for handling exception:
  - » Catch the exception in the method `evaluate` (where it is thrown)
  - » Declare `evaluate` as throwing the exception and catch it in `doCalculation`
  - » Declare both `evaluate` and `doCalculation` as throwing the exception and handle it in `main`
- Asks user to re-enter the calculation, so it uses the third option
- Also includes an exception for division by zero

Listing 9.10 The UnknownOpException Class - UnknownOpException.java

```
// Listing 9.10 The UnknownOpException Class

public class UnknownOpException extends Exception
{
   public UnknownOpException( )
   {
      super("UnknownOpException");
   }

   public UnknownOpException(char op)
   {
      super(op + " is an unknown operator.");
   }

   public UnknownOpException(String message)
   {
      super(message);
   }
}
```

Listing 9.11 The Unexceptional Cases - PrelimCalculator.java

- » Without exception handling
- » Ex) /0.0 …. DivideByZeroException

```
import java.util.Scanner;

/**
 PRELIMINARY VERSION without exception handling.
 Simple line-oriented calculator program. The class
 can also be used to create other calculator programs.
*/
public class PrelimCalculator
{
   private double result;
   private double precision = 0.0001; // Numbers this close to zero are
                                      // treated as if equal to zero.

   public static void main(String[] args) throws DivideByZeroException,
                                                  UnknownOpException
   {
      PrelimCalculator clerk = new PrelimCalculator( );
```

```
System.out.println("Calculator is on.");
      System.out.print("Format of each line: ");
      System.out.println("operator space number");
      System.out.println("For example: + 3");
      System.out.println("To end, enter the letter e.");
      clerk.doCalculation();

      System.out.println("The final result is " +
                                                    clerk.getResult( ));
      System.out.println("Calculator program ending.");
   }
   public PrelimCalculator( )
   {
      result = 0;
   }
   public void reset( )
   {
      result = 0;
   }

   public void setResult(double newResult)
   {
      result = newResult;
   }
```

```
public double getResult( )
  {
     return result;
  }

  public void doCalculation( ) throws DivideByZeroException,
                                      UnknownOpException
  {
     Scanner keyboard = new Scanner(System.in);
     boolean done = false;
     result = 0;
     System.out.println("result = " + result);

     while (!done)
     {
        char nextOp = (keyboard.next( )).charAt(0);
         if ((nextOp == 'e') || (nextOp == 'E'))
             done = true;
         else
         {
             double nextNumber = keyboard.nextDouble( );
             result = evaluate(nextOp, result, nextNumber);
             System.out.println("result " + nextOp + " " +
                                 nextNumber + " = " + result);
             System.out.println("updated result = " + result);
         }
     }
  }
```

```
/**
   Returns n1 op n2, provided op is one of '+', '-', '*',or '/'.
   Any other value of op throws UnknownOpException.
  */
  public double evaluate(char op, double n1, double n2)
          throws DivideByZeroException, UnknownOpException
  {
     double answer;
     switch (op)
     {
        case '+':
           answer = n1 + n2;
           break;
        case '-':
           answer = n1 - n2;
           break;
        case '*':
           answer = n1 * n2;
           break;
        case '/':
           if ((-precision < n2) && (n2 < precision))
              throw new DivideByZeroException( );
           answer = n1 / n2;
           break;
        default:
           throw new UnknownOpException(op);
     }
     return answer;
  }
}
```

C:₩Windows₩system32₩cmd.exe

```
Calculator is on.
Format of each line: operator space number
For example: + 3
To end, enter the letter e.
result = 0.0
+ 4
result + 4.0 = 4.0
updated result = 4.0
/ 0
Exception in thread "main" DivideByZeroException: Dividing by Zero!
        at PrelimCalculator.evaluate(PrelimCalculator.java:93)
        at PrelimCalculator.doCalculation(PrelimCalculator.java:64)
        at PrelimCalculator.main(PrelimCalculator.java:24)
계속하려면 아무 키나 누르십시오 . . .
```

C:₩Windows₩system32₩cmd.exe

```
Calculator is on.
Format of each line: operator space number
For example: + 3
To end, enter the letter e.
result = 0.0
+ 4
result + 4.0 = 4.0
updated result = 4.0
# 2
Exception in thread "main" UnknownOpException: # is an unknown operator.
        at PrelimCalculator.evaluate(PrelimCalculator.java:97)
        at PrelimCalculator.doCalculation(PrelimCalculator.java:64)
        at PrelimCalculator.main(PrelimCalculator.java:24)
계속하려면 아무 키나 누르십시오 . . .
```

Listing 9.12 The Complete Line-Oriented Calculator - Calculator.java

```
import java.util.Scanner;

/**
 Simple line-oriented calculator program. The class
 can also be used to create other calculator programs.
*/
public class Calculator
{
   private double result;
   private double precision = 0.0001; // Numbers this close to zero are
                                      // treated as if equal to zero.

   public static void main(String[] args)
   {
      Calculator clerk = new Calculator( );
```

```
    try
    {
        System.out.println("Calculator is on.");
        System.out.print("Format of each line: ");
        System.out.println("operator space number");
        System.out.println("For example: + 3");
        System.out.println("To end, enter the letter e.");
        clerk.doCalculation();
    }
    catch(UnknownOpException e)
    {
        clerk.handleUnknownOpException(e);
    }
    catch(DivideByZeroException e)
    {
        clerk.handleDivideByZeroException(e);
    }
    System.out.println("The final result is " +
                                                clerk.getResult( ));
    System.out.println("Calculator program ending.");
}

public Calculator( )
{
    result = 0;
}
```

```
public void reset( )
 {
    result = 0;
 }

 public void setResult(double newResult)
 {
    result = newResult;
 }

 public double getResult( )
 {
    return result;
 }
```

```
/**
   The heart of a calculator. This does not give
   instructions. Input errors throw exceptions.
  */
  public void doCalculation( ) throws DivideByZeroException,
                                      UnknownOpException
  {
     Scanner keyboard = new Scanner(System.in);

     boolean done = false;
     result = 0;
     System.out.println("result = " + result);
     while (!done)
     {
        char nextOp = (keyboard.next( )).charAt(0);
         if ((nextOp == 'e') || (nextOp == 'E'))
             done = true;
         else
         {
             double nextNumber = keyboard.nextDouble( );
             result = evaluate(nextOp, result, nextNumber);
             System.out.println("result " + nextOp + " " +
                                 nextNumber + " = " + result);
             System.out.println("updated result = " + result);
         }
     }
  }
```

```
/**
   Returns n1 op n2, provided op is one of '+', '-', '*',or '/'.
   Any other value of op throws UnknownOpException.
  */
  public double evaluate(char op, double n1, double n2)
            throws DivideByZeroException, UnknownOpException
  {
     double answer;
     switch (op)
     {
        case '+':
           answer = n1 + n2;
           break;
        case '-':
           answer = n1 - n2;
           break;
        case '*':
           answer = n1 * n2;
           break;
        case '/':
           if ((-precision < n2) && (n2 < precision))
              throw new DivideByZeroException( );
           answer = n1 / n2;
           break;
        default:
           throw new UnknownOpException(op);
     }
     return answer;
  }
```

```
public void handleDivideByZeroException(DivideByZeroException e)
 {
    System.out.println("Dividing by zero.");
    System.out.println("Program aborted");
    System.exit(0);
 }

 public void handleUnknownOpException(UnknownOpException e)
 {
    System.out.println(e.getMessage( ));
    System.out.println("Try again from the beginning:");

    try
    {
       System.out.print("Format of each line: ");
       System.out.println("operator number");
       System.out.println("For example: + 3");
       System.out.println("To end, enter the letter e.");
       doCalculation( );
    }
```

```
        catch(UnknownOpException e2)
        {
            System.out.println(e2.getMessage( ));
            System.out.println("Try again at some other time.");
            System.out.println("Program ending.");
            System.exit(0);
        }
        catch(DivideByZeroException e3)
        {
            handleDivideByZeroException(e3);
        }
    }
}
```

C:₩Windows₩system32₩cmd.exe

```
Calculator is on.
Format of each line: operator space number
For example: + 3
To end, enter the letter e.
result = 0.0
+ 80
result + 80.0 = 80.0
updated result = 80.0
- 2
result - 2.0 = 78.0
updated result = 78.0
% 4
% is an unknown operator.
Try again from the beginning:
Format of each line: operator number
For example: + 3
To end, enter the letter e.
result = 0.0
+ 80
result + 80.0 = 80.0
updated result = 80.0
- 2
result - 2.0 = 78.0
updated result = 78.0
* 0.04
result * 0.04 = 3.12
updated result = 3.12
e
The final result is 3.12
Calculator program ending.
계속하려면 아무 키나 누르십시오 . . .
```

끝