Self-Test Exercises (continued)

8. In the code given in Self-Test Exercise 3, what is the `catch` block?
9. In the code given in Self-Test Exercise 3, what is the `catch` block parameter?
10. Is the following legal?

```
Exception exceptionObject =
                        new Exception("Oops!");
```

11. Is the following legal?

```
Exception exceptionObject =
                        new Exception("Oops!");
throw exceptionObject;
```

Defining Exception Classes

A `throw` statement can throw an exception object of any exception class. A common thing to do is to define an exception class whose objects can carry the precise kinds of information you want thrown to the `catch` block. An even more important reason for defining a specialized exception class is so that you can have a different type to identify each possible kind of exceptional situation.

Every exception class you define must be a derived class of some already defined exception class. An exception class can be a derived class of any exception class in the standard Java libraries or of any exception class that you have already successfully defined. Our examples will be derived classes of the class `Exception`.

constructors

When defining an exception class, the constructors are the most important members. Often there are no other members, other than those inherited from the base class. For example, in Display 9.4, we have defined an exception class called `DivisionByZeroException` whose only members are a no-argument constructor and a constructor with one `String` parameter. In most cases, these two constructors are all the exception class definition contains. However, the class does inherit all the methods of the class `Exception`.[1] In particular, the class `DivisionByZeroException` inherits the method `getMessage`, which returns a string message. In the no-argument constructor, this string message is set with the following, which is the first line in the no-argument constructor definition:

```
super("Division by Zero!");
```

This is a call to a constructor of the base class `Exception`. As we have already noted, when you pass a string to the constructor for the class `Exception`, it sets the value

[1]Some programmers would prefer to derive the `DivisionByZeroException` class from the predefined class `ArithmeticException`, but that would make it a kind of exception that you are not required to catch in your code, so you would lose the help of the compiler in keeping track of uncaught exceptions. For more details, see the subsection "Exceptions to the Catch or Declare Rule" later in this chapter. If this footnote does not make sense to you, you can safely ignore it.

of a `String` instance variable that can later be recovered with a call to `getMessage`. The method `getMessage` is an ordinary accessor method of the class `Exception`. The class `DivisionByZeroException` inherits this `String` instance variable as well as the accessor method `getMessage`.

For example, in Display 9.5, we give a sample program that uses this exception class. The exception is thrown using the no-argument constructor, as follows:

```
throw new DivisionByZeroException();
```

Display 9.4 **A Programmer-Defined Exception Class**

```
1  public class DivisionByZeroException extends Exception
2  {
3      public DivisionByZeroException()
4      {
5          super("Division by Zero!");
6      }

7      public DivisionByZeroException(String message)
8      {
9          super(message);
10     }
11 }
```

You can do more in an exception constructor, but this form is common.

***super** is an invocation of the constructor for the base class **Exception**.*

Display 9.5 **Using a Programmer-Defined Exception Class** (part 1 of 3)

```
1  import java.util.Scanner;

2  public class DivisionDemoFirstVersion
3  {

4      public static void main(String[] args)
5      {
6          try
7          {
8              Scanner keyboard = new Scanner(System.in);

9              System.out.println("Enter numerator:");
10             int numerator = keyboard.nextInt();
11             System.out.println("Enter denominator:");
12             int denominator = keyboard.nextInt();
```

We will present an improved version of this program later in this chapter in Display 9.10.

(continued)

Display 9.5 **Using a Programmer-Defined Exception Class** (part 2 of 3)

```
13              if (denominator == 0)
14                  throw new DivisionByZeroException();

15              double quotient = numerator/(double)denominator;
16              System.out.println(numerator + "/"
17                                 + denominator
18                                 + " = " + quotient);
19          }
20          catch (DivisionByZeroException e)
21          {
22              System.out.println(e.getMessage());
23              secondChance();
24          }

25          System.out.println("End of program.");
26      }

27      public static void secondChance()
28      {
29          Scanner keyboard = new Scanner(System.in);
30          System.out.println("Try again:");
31          System.out.println("Enter numerator:");
32          int numerator = keyboard.nextInt();
33          System.out.println("Enter denominator:");
34          System.out.println("Be sure the denominator is not zero.");
35          int denominator = keyboard.nextInt();
36
37          if (denominator == 0)
38          {
39              System.out.println("I cannot do division by zero.");
40              System.out.println("Aborting program.");
41              System.exit(0);
42          }

43          double quotient = ((double)numerator)/denominator;
44          System.out.println(numerator + "/"
45                                       + denominator
46                                       + " = " + quotient);
47      }
48  }
```

Sometimes it is better to handle an exceptional case without throwing an exception.

Display 9.5 **Using a Programmer-Defined Exception Class** (part 3 of 3)

Sample Dialogue 1

```
Enter numerator:
11
Enter denominator:
5
11/5 = 2.2
End of program.
```

Sample Dialogue 2

```
Enter numerator:
11
Enter denominator:
0
Division by Zero!
Try again.
Enter numerator:
11
Enter denominator:
Be sure the denominator is not zero.
5
11/5 = 2.2
End of program.
```

Sample Dialogue 3

```
Enter numerator:
11
Enter denominator:
0
Division by Zero!
Try again.
Enter numerator:
11
Enter denominator:
Be sure the denominator is not zero.
0
I cannot do division by zero.
Aborting program.
```

Multiple `catch` Blocks

A `try` block can potentially throw any number of exception values, and they can be of differing types. In any one execution of the `try` block, at most one exception will be thrown (since a `throw` statement ends the execution of the `try` block), but different types of exception values can be thrown on different occasions when the `try` block is executed. Each `catch` block can only catch values of the exception class type given in the `catch` block heading. However, you can catch exception values of differing types by placing more than one `catch` block after a `try` block. For example, the program in Display 9.8 has two `catch` blocks after its `try` block. The class `NegativeNumberException`, which is used in that program, is given in Display 9.9.

Display 9.8 **Catching Multiple Exceptions** (part 1 of 2)

```
1  import java.util.Scanner;

2  public class MoreCatchBlocksDemo
3  {
4     public static void main(String[] args)
5     {
6         Scanner keyboard = new Scanner(System.in);
7
8         try
9         {
10            System.out.println("How many pencils do you have?");
11            int pencils = keyboard.nextInt();

12            if (pencils < 0)
13                 throw new NegativeNumberException("pencils");

14            System.out.println("How many erasers do you have?");
15            int erasers = keyboard.nextInt();
16            double pencilsPerEraser;

17            if (erasers < 0)
18                 throw new NegativeNumberException("erasers");
19            else if (erasers != 0)
20                 pencilsPerEraser = pencils/(double)erasers;
21            else
22                 throw new DivisionByZeroException();

23            System.out.println("Each eraser must last through "
24                + pencilsPerEraser + " pencils.");
25        }
26        catch(NegativeNumberException e)
27        {
28            System.out.println("Cannot have a negative number of "
29                + e.getMessage());
30        }
31        catch(DivisionByZeroException e)
32        {
33            System.out.println("Do not make any mistakes.");
```

Display 9.8 **Catching Multiple Exceptions** (part 2 of 2)

```
34          }

35          System.out.println("End of program.");
36      }
37  }
```

Sample Dialogue 1

```
How many pencils do you have?
5
How many erasers do you have?
2
Each eraser must last through 2.5 pencils
End of program.
```

Sample Dialogue 2

```
How many pencils do you have?
-2
Cannot have a negative number of pencils
End of program.
```

Sample Dialogue 3

```
How many pencils do you have?
5
How many erasers do you have?
0
Do not make any mistakes.
End of program.
```

PITFALL: Catch the More Specific Exception First

When catching multiple exceptions, the order of the `catch` blocks can be important. When an exception is thrown in a `try` block, the `catch` blocks are examined in order, and the first one that matches the type of the exception thrown is the one that is executed. Thus, the following ordering of `catch` blocks would not be good:

```
catch (Exception e)
{
        .
        .
        .
}
```

(continued)

Display 9.10 Use of a `throws` Clause (part 1 of 2)

```
1  import java.util.Scanner;

2  public class DivisionDemoSecondVersion
3  {
4      public static void main(String[] args)
5      {
6          Scanner keyboard = new Scanner(System.in);

7          try
8          {
9              System.out.println("Enter numerator:");
10             int numerator = keyboard.nextInt();
11             System.out.println("Enter denominator:");
12             int denominator = keyboard.nextInt();
13             double quotient = safeDivide(numerator, denominator);
14             System.out.println(numerator + "/"
15                                        + denominator
16                                        + " = " + quotient);
17         }
18         catch (DivisionByZeroException e)
19         {
20             System.out.println(e.getMessage());
21             secondChance();
22         }
23
24         System.out.println("End of program.");
25     }
26

27     public static double safeDivide(int top, int bottom)
28                           throws DivisionByZeroException
29     {
30         if (bottom == 0)
31            throw new DivisionByZeroException();

32         return top/(double)bottom;
33     }
```

(continued)

Display 9.10 Use of a `throws` Clause (part 2 of 2)

```
34      public static void secondChance()
35      {
36          Scanner keyboard = new Scanner(System.in);
37
38          try
39          {
40              System.out.println("Enter numerator:");
41              int numerator = keyboard.nextInt();
42              System.out.println("Enter denominator:");
43              int denominator = keyboard.nextInt();

44              double quotient = safeDivide(numerator, denominator);
45              System.out.println(numerator + "/"
46                                          + denominator
47                                          + " = " + quotient);
48          }
49          catch(DivisionByZeroException e)
50          {
51              System.out.println("I cannot do division by zero.");
52              System.out.println("Aborting program.");
53              System.exit(0);
54          }
55      }
56  }
```

The input/output dialogues are identical to those for the program in Display 9.5.

Declaring Exceptions in a `throws` Clause

`throws` clause

declaring an exception

If a method does not catch an exception, then (in most cases) it must at least warn programmers that any invocation of the method might possibly throw an exception. This warning is called a *throws clause*, and including an exception class in a `throws` clause is called **declaring the exception**. For example, a method that might possibly throw a `DivisionByZeroException` and that does not catch the exception would have a heading similar to the following:

```
public void sampleMethod()throws DivisionByZeroException
```

`throws` clause

The part `throws DivisionByZeroException` is a **`throws` clause** stating that an invocation of the method `sampleMethod` might throw a `DivisionByZeroException`.

If there is more than one possible exception that can be thrown in the method definition, then the exception types are separated by commas, as illustrated in what follows:

```
public void sampleMethod()
            throws DivisionByZeroException, SomeOtherException
```