Chapter 9

Exception Handling

- 9.1 Basic Exception Handling
- 9.2 Defining Your Own Exception
- 9.3 More about Exception Classes
- 9.4 Graphics Supplement (Optional)

Objectives

- 1) Become familiar with the notion of exception handling
- 2) Learn Java syntax for exception handling
- 3) Develop the ability to use exception handling effectively in your own classes and programs

Exception Handling Overview

- A way of organizing a program into sections for the normal case and the exceptional case
  - » exception examples:
    division by zero
    incorrect type of input

- A way of implementing programs **incrementally**
  - » write & debug the code for first
  - » add code for the later

- Simplifies development, testing, debugging and maintenance
  - » errors are easier to isolate

9.1 Basic Exception Handling - Some Terminology

- ***an exception***: either Java itself or your code signals when something unusual happens
- ***an exception***: responding to an exception by executing a part of the program specifically written for the exception
  - » also called *an exception*

`try-throw-catch` Threesome

Basic code organization:

```
try
{
    <code to try>
    if(test condition)
      throw new Exception("Message to display");
    <more code>
}
catch(Exception e)
{
    <exception handling code>
}

<possibly more code>
```

- The normal case is handled **in a** [redacted]
- The exceptional case is handled **in** [redacted]
- The catch block takes a parameter of type [redacted]
  - » it is called the *`catch`-block parameter*
  - » `e` is a commonly used name for it
- If an exception is *thrown*, execution in the `try` block ends and control passes to the `catch` block(s) after the `try` block

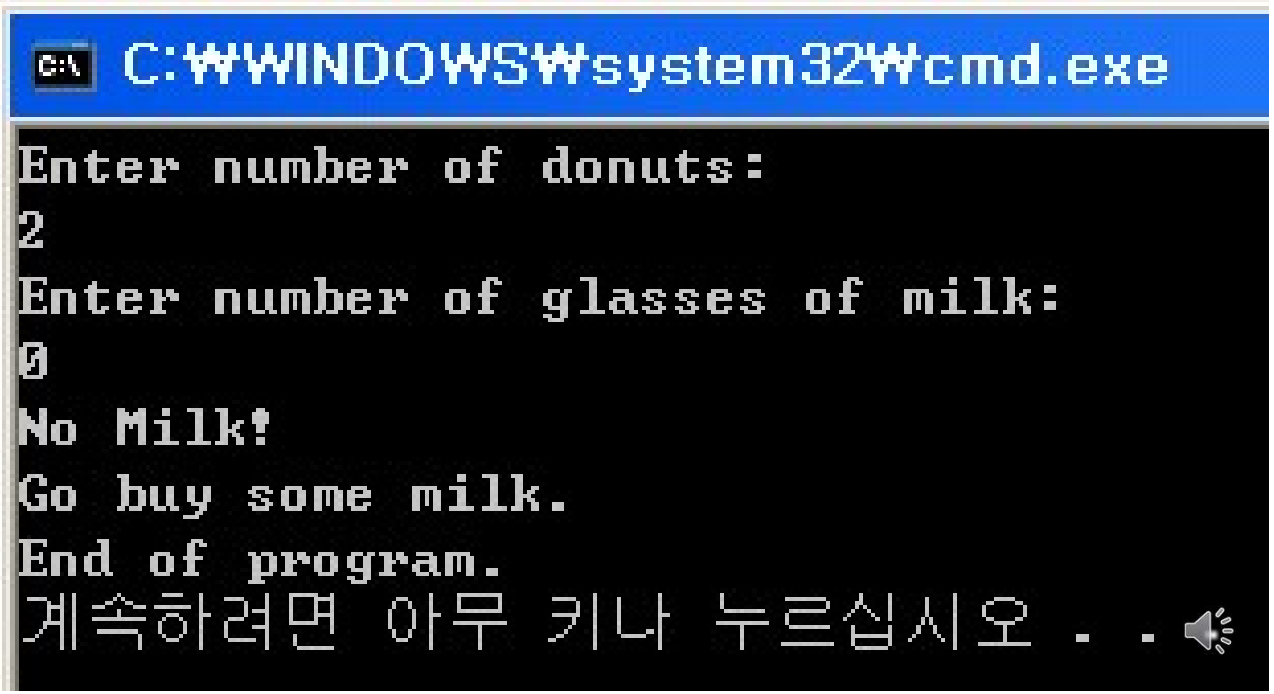
Listing 9.1. One Way to Deal with a Problem Situation - GotMilk.java

```
import java.util.Scanner;

public class GotMilk
{
   public static void main(String[] args)
   {
      Scanner keyboard = new Scanner(System.in);

      System.out.println("Enter number of donuts:");
      int donutCount = keyboard.nextInt( );
      System.out.println("Enter number of glasses of milk:");
      int milkCount = keyboard.nextInt( );
```

```
        if (milkCount < 1)
        {
            System.out.println("No milk!");
            System.out.println("Go buy some milk.");
        }
        else
        {
            double donutsPerGlass = donutCount / (double)milkCount;
            System.out.println(donutCount + " donuts.");
            System.out.println(milkCount + " glasses of milk.");
            System.out.println("You have " + donutsPerGlass +
                               " donuts for each glass of milk.");
        }
        System.out.println("End of program.");
    }
}
```



Lising 9.2. An Example of Exception Handling - ExceptionDemo.java

```
import java.util.Scanner;

public class ExceptionDemo
{
   public static void main(String[] args)
   {
      Scanner keyboard = new Scanner(System.in);
```

```
        try
        {
            System.out.println("Enter number of donuts:");
            int donutCount = keyboard.nextInt( );
            System.out.println("Enter number of glasses of milk:");
            int milkCount = keyboard.nextInt( );
            if (milkCount < 1)
              throw new Exception("Exception: No milk!");
            double donutsPerGlass = donutCount / (double)milkCount;
            System.out.println(donutCount + " donuts.");
            System.out.println(milkCount + " glasses of milk.");
            System.out.println("You have " + donutsPerGlass +
                                  " donuts for each glass of milk.");
        }
        catch(Exception e)
        {
            System.out.println(e.getMessage());
            System.out.println("Go buy some milk.");
        }
        System.out.println("End of program.");
    }
}
```

```
C:₩WINDOWS₩system32₩cmd.exe
Enter number of donuts:
2
Enter number of glasses of milk:
0
Exception: No Milk!
Go buy some milk.
End of program.
계속하려면 아무 키나 누르십시오 . . .
```

`try-throw-catch` Program Flow

Try block

Statements execute up to the conditional `throw` statement

If the condition is `true` the exception is thrown

- » control passes to the `catch` block(s) after the `try` block

Else the condition is `false`

- » the exception is not thrown
- » the remaining statements in the `try` block (those following the conditional throw) are executed

Catch block

Executes if an exception is thrown

- » may terminate execution with **`exit` statement**
- » if it does not exit, execution resumes after the `catch` block

Statements after the Catch block

- » Executes if either the exception is not thrown or if it is thrown but the `catch` block does not exit

An Example of Exception Handling

`ExceptionDemo`

try and catch blocks

`try` block

`throw` statement

`catch` block

```
try
{
    System.out.println("Enter number of donuts:");
    int donutCount = keyboard.nextInt( );

    System.out.println("Enter number of glasses of milk:");
    int milkCount = keyboard.nextInt( );

    if (milkCount < 1)
      throw new Exception("Exception: No Milk!");

    donutsPerGlass = donutCount/(double)milkCount;
    System.out.println(donutCount + " donuts.");
    System.out.println(milkCount + " glasses of milk.");
    System.out.println("You have " + donutsPerGlass
                + " donuts for each glass of milk.");
}
```

```
catch(Exception e)
{
    System.out.println(e.getMessage());
    System.out.println("Go buy some milk.");
    System.out.println("Program aborted.");
    System.exit(0);
}
```

Listing 9.3

Flow of Control with Exception Handling

```
try
{
    System.out.println("Ente
    int donutCount = keyboar
    System.out.println("Ente
    int milkCount = keyboard.nextInt( );
    if (milkCount < 1)
        throw new Exception("Exception: No Milk!");
    donutsPerGlass = donutCount/(double)milkCount;
    System.out.println(donutCount + "
    System.out.println(milkCount + " g
    System.out.println("You have " + d
}
catch(Exception e)
{
    System.out.println(e.getMessage());
    System.out.println("Go buy some milk.");
}
System.out.println("End of program.");
```

Assume user enters a positive number for number of glasses, so no exception is thrown.

Not executed when there's no exception thrown.

Main method from `Exception-Demo` program

Listing 9.4

Flow of Control with Exception Handling

```
try
{
    System.out.println("Ente
    int donutCount = keyboar
    System.out.println("Ente
    int milkCount = keyboard.nextInt( );
    if (milkCount < 1)
        throw new Exception("Exception: No Milk!");
    donutsPerGlass = donutCount/(double)milkCount;
    System.out.println(donutCount + " donuts.");
    System.out.println(milkCount + " glasses of milk.");
    System.out.println("You have " + donutsPerGlass);
}
catch(Exception e)
{
    System.out.println(e.getMessage());
    System.out.println("Go buy some milk.");
}
System.out.println("End of program.");
```

Assume user enters zero or a negative number for number of glasses, so an exception is thrown.

Not executed when an exception is thrown

Main method from `Exception-Demo` program

More about the **`catch`**-Block

- Although it may look similar to a method definition

The `catch`-block is a method definition!

- Every `Exception` has **a method**
  - » it retrieves the string given to the exception object when it was thrown, e.g.

```
throw new Exception("This message is retrieved");
```

- A `catch`-block applies only to an immediately preceding `try` block
  - » if no exception is thrown the catch block is ignored

- `Catch Block Parameter : Catch(Exception e)`
  - » What kind of exception the catch block can catch .

An Exception is

- Throw new Exception(“Illegal character on line 57.”);
  - » it creates an object that has a message.
  - » the message is “Illegal character on line 57.”

  - » Exception e = new Exception(“Illegal character on line 57.”);

Predefined Exception Classes

- `Exception` is the root class of all exceptions
- Many predefined classes throw exceptions
  - » the documentation or interface will tell you
  - » the exceptions thrown are often also predefined
- Some common predefined exceptions:
  - » `IOException`
  - » `ClassNotFoundException`, and
  - » `FileNotFoundException`

Code Organization when Using an Object that May Throw an Exception

```
Sample object = new SampleClass();
try
{
   <Possibly some code>
   object.doStuff;//may throw IOException
   <Possibly some more code>
}
catch(IOException e)
{
   <Code to handle the IOException, probably
  including this line:>
   System.out.println(e.getMessage());
}
```

- Predefined exceptions usually include a meaningful message that is retrieved with `getMessage`

ArrayIndexOutOfBoundsException

- thrown if your program attempts to use an array index that is out of bounds (too big or a negative number)
- does not need to be caught or accounted for in any way
- normally is not caught in a catch block
- functions more like a run-time error than a regular exception

끝