Exception Handling

Chapter 9

*JAVA: An Introduction to Problem Solving & Programming, 6*th Ed. By Walter Savitch
ISBN 0132162709

Objectives

- Describe the notion of exception handling
- React correctly when certain exceptions occur
- Use Java's exception-handling facilities effectively in classes and programs

Exceptions

- Download from SavitchSrc link ch09:
  - **GotMilk.java**
  - **ExceptionDemo.java**
  - **DivideByZeroException.java**
  - **DivideByZeroDemo.java**
  - **DoDivision.java**
  - **TwoCatchesDemo.java**
  - **NegativeNumberException.java**

Exceptions in Java

- An exception is an object
  - Signals the occurrence of unusual event during program execution
- Throwing an exception
  - Creating the exception object
- Handling the exception
  - Code that detects and deals with the exception

Exceptions in Java

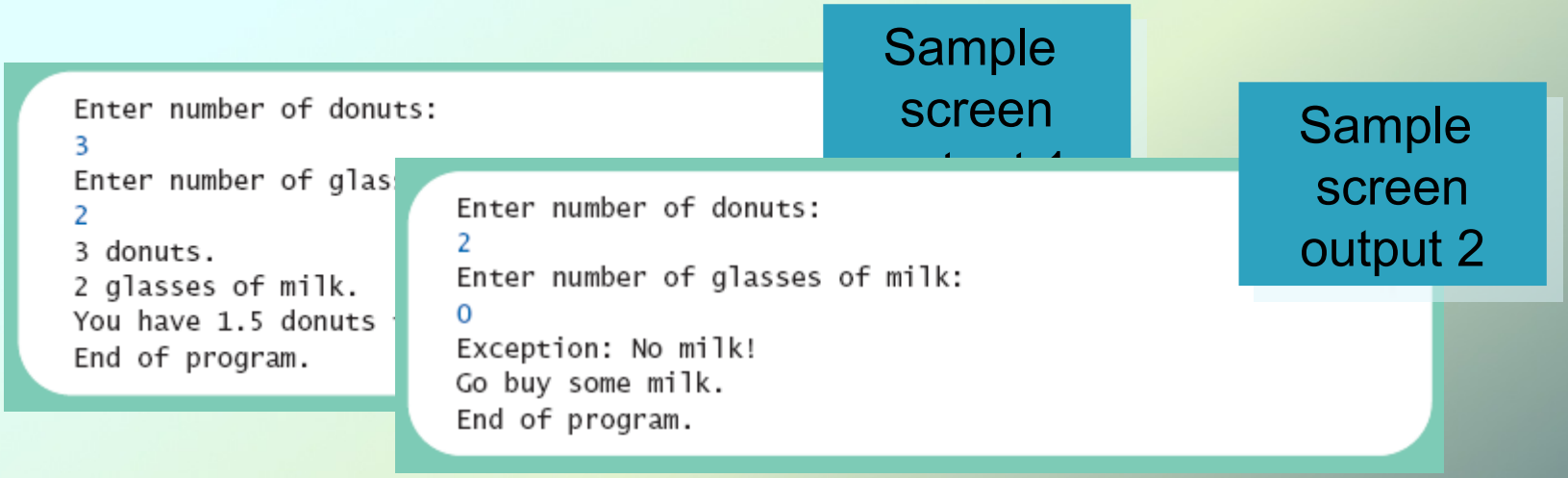
- Consider a program to assure us of a sufficient supply of milk
- View **GotMilk.java**

```
Enter number of donuts:
2
Enter number of glasses of milk:
0
No milk!
Go buy some milk.
End of program.
```

Sample screen output

Exceptions in Java

- Now we revise the program to use exception-handling
- View **`ExceptionDemo.java`**



Exceptions in Java

- Note `try` block
  - Contains code where something could possibly go wrong
  - If it does go wrong, we *throw an exception*
- Note `catch` block
  - When exception thrown, `catch` block begins execution
  - Similar to method with parameter
  - Parameter is the thrown object

Exceptions in Java

```
import java.util.*;

public class ExceptionDemo
{
    public static void main(String[] args)
    {
        int donutCount, milkCount;
        double donutsPerGlass;
        Scanner keyboard = new Scanner(System.in);

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

            System.out.println("Enter number of glasses of milk:");
            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("End of program.");
    }
}
```

These arrows assume that the user inputs a positive number for the number of glasses of milk.

milkCount is positive, so an exception is NOT thrown here.

This code is NOT executed.

Display 8.3

Flow of Control with Exception Handling

- Note flow of control when no exception is thrown
- View **`ExceptionDemo`**

Sample screen output with no exception

```
Enter number of donuts:
3
Enter number of glasses of milk:
2
3 donuts.
2 glasses of milk.
You have 1.5 donuts for each glass of milk.
End of program.
```

Exceptions in Java

```
import java.util.*;

public class ExceptionDemo
{
    public static void main(String[] args)
    {
        int donutCount, milkCount;
        double donutsPerGlass;
        Scanner keyboard = new Scanner(System.in);

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

            System.out.println("Enter number of glasses of milk:");
            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("End of program.");
    }
}
```

These arrows assume that the user inputs zero for the number of glasses of milk, and so an exception is thrown.

`milkCount` *is zero or negative, so an exception is thrown here.*

This code is NOT executed.

Display 8.3

Flow of Control with Exception Handling

- Note flow of control when exception IS thrown
- View **`ExceptionDemo`**

Sample screen output when exception is thrown

```
Enter number of donuts:
2
Enter number of glasses of milk:
0
Exception: No milk!
Go buy some milk.
End of program.
```

Predefined Exception Classes

- Java has predefined exception classes within Java Class Library
  - Can place method invocation in `try` block
  - Follow with `catch` block for this type of exception
- Example classes
  - `BadStringOperationException`
  - `ClassNotFoundException`
  - `IOException`
  - `NoSuchMethodException`

Predefined Exception Classes

- Example code

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

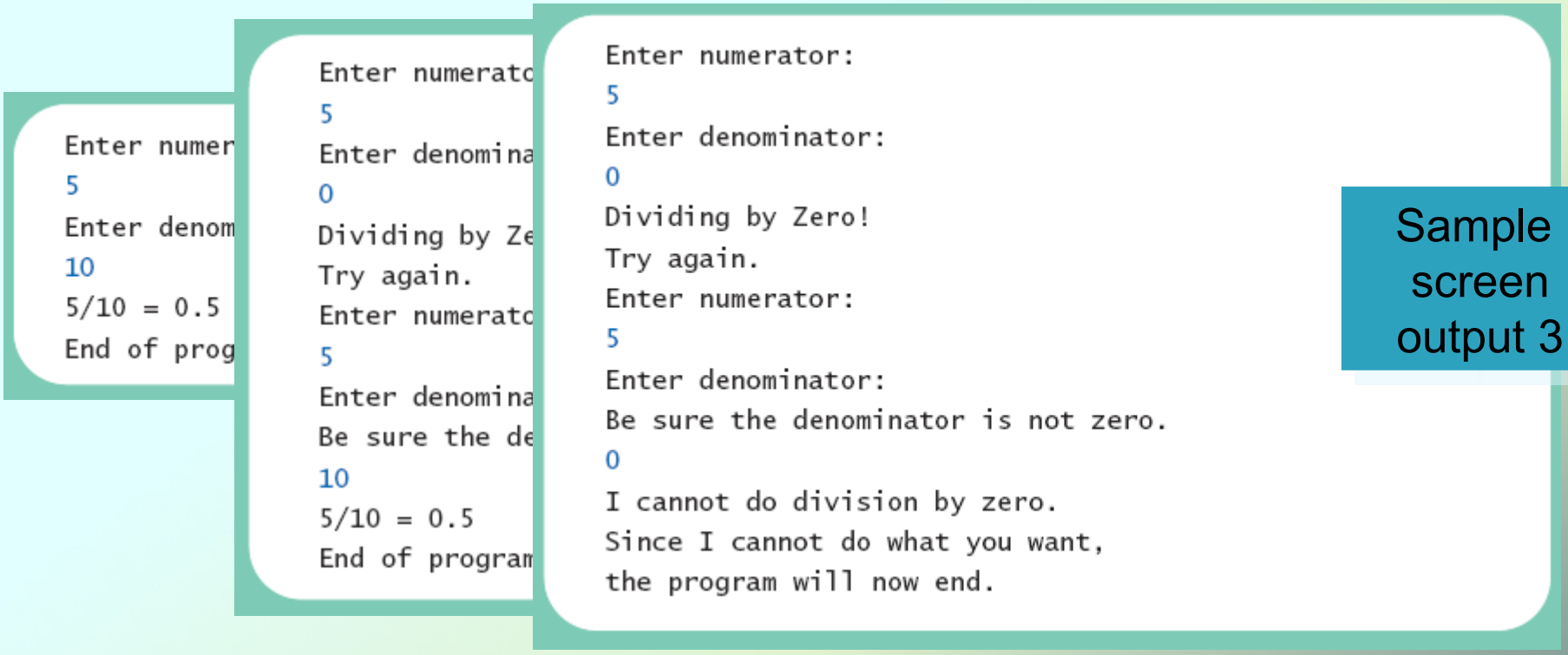
Defining Your Own Exception Classes

- Must be derived class of some predefined exception class
  - Use classes derived from class **`Exception`**
- View **`DivideByZeroException.java`**

  `extends Exception`
- View **`DivideByZeroDemo.java`**

Defining Your Own Exception Classes

- Different runs of the program



Sample screen output 3

Defining Your Own Exception Classes

- Note method `getMessage` defined in exception classes
  - Returns string passed as argument to constructor
  - If no actual parameter used, default message returned
- The type of an object is the name of the exception class

Defining Your Own Exception Classes

Guidelines

- Use the **`Exception`** as the base class
- Define at least two constructors
  - Default, no parameter
  - With **`String`** parameter
- Start constructor definition with call to constructor of base class, using **`super`**
- Do not override inherited **`getMessage`**

More About Exception Classes: Outline

- Declaring Exceptions (Passing the Buck)
- Kinds of Exceptions
- Errors
- Multiple Throws and Catches
- The `finally` Block
- Rethrowing an Exception

Declaring Exceptions

- Consider method where code throws exception
  - May want to handle immediately
  - May want to delay until something else is done
- Method that does not catch an exception
  - Notify programmers with **`throws`** clause
  - Programmer then given responsibility to handle exception

Declaring Exceptions

- Note syntax for throws clause

```
public Type Method_Name(Parameter_List) throws List_Of_Exceptions
Body_Of_Method
```

- Note distinction
  - Keyword **`throw`** used to throw exception
  - Keyword **`throws`** used in method heading to declare an exception

Declaring Exceptions

- If a method throws exception and exception not caught inside the method
  - Method ends immediately after exception thrown
- A throws clause in overriding method
  - Can declare fewer exceptions than declared
  - But not more
- View **`DoDivision.java`**

Kinds of Exceptions

- In most cases, exception is caught …
  - In a **`catch`** block … or
  - Be declared in **`throws`** clause
- But Java has exceptions you do not need to account for
- Categories of exceptions
  - Checked exceptions
  - Unchecked exceptions

Kinds of Exceptions

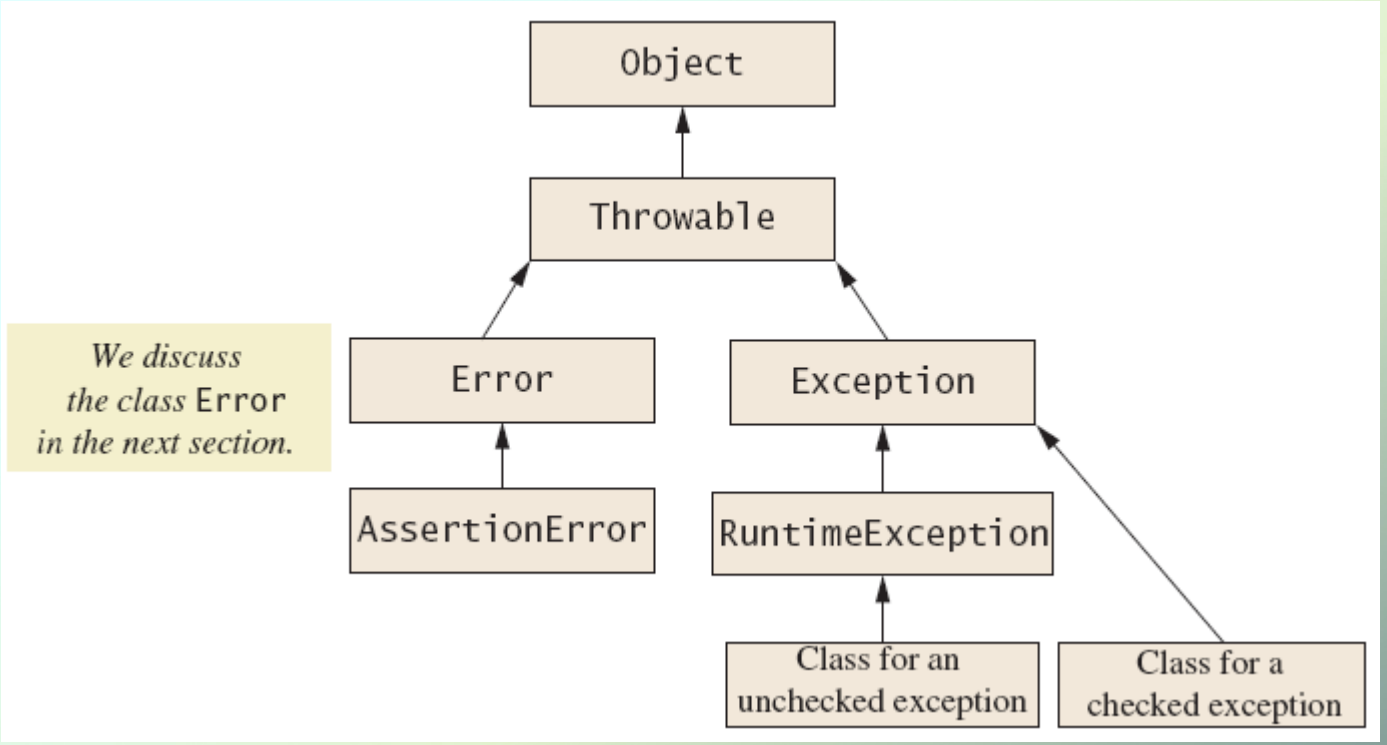
- *Checked* exception
  - Must be caught in **`catch`** block
  - Or declared in **`throws`** clause
- *Unchecked* exception
  - Also called *run-time*
  - Need not be caught in **`catch`** block or declared in **`throws`**
  - Exceptions that indicate coding problems exist, should be fixed

Kinds of Exceptions

- Examples why unchecked exceptions are thrown
  - Attempt to use array index out of bounds
  - Division by zero
- Uncaught runtime exception terminates program execution

Kinds of Exceptions

- Figure 9.1 Hierarchy of the predefined exception classes



Multiple Throws and Catches

- A try block can throw any number of exceptions of different types
- Each catch block can catch exceptions of only one type
  - Order of catch blocks matter
- View **`TwoCatchesDemo.java`**
- View **`NegativeNumberException.java`**

Multiple Throws and Catches

- Note multiple sample runs

```
Enter number of widgets produced:
1000
How many were defective?
500
One in every 2.0 widgets is defective.
End o
```

Sample screen output 1

```
Enter number of widgets produced:
-10
Cannot have a negative number of widgets
End of prog
```

Sample screen

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

Sample screen output 2

Multiple Throws and Catches

- Exceptions can deal with invalid user input
- To handle an exception thrown by a method
  - It does not matter where in the method the **`throw`** occurs
- Use of **`throw`** statement should be reserved for cases where it is unavoidable
- Separate methods for throwing and catching of exceptions
- Nested try-catch blocks rarely useful

The `finally` Block

- Possible to add a `finally` block after sequence of `catch` blocks
- Code in `finally` block executed
  - Whether or not execution thrown
  - Whether or not required `catch` exists

Rethrowing an Exception

- Legal to throw an exception within a **`catch`** block
- Possible to use contents of **`String`** parameter to **`throw`** same or different type exception

Exercise

- Write a class definition called **`EmailAddress`**:
  - constructor that takes an address
    - throws **`IllegalEmailException`** if the address does not contain '@' (recall **`String.indexOf(char)`**)
  - method **`getHost`**
    - everything after the '@'
  - method **`getName`**
    - everything before the '@'
- Write a demo program