Object Oriented Programming

Introduction to Java

Ch. 9. Exception Handling

Dept. of Software, Gachon University
Ahyoung Choi, Spring

Question

When a program runs into a runtime error, the program terminates abnormally.

How can you handle the runtime error so that the program can continue to run or terminate gracefully?

Approach 1 : Traditional Methods of Handling Errors

- In most procedural languages, the standard way of indicating an error condition is by returning an error code.
- The calling code typically did one of the following:
  - Testing the error code and taking the appropriate action.
  - Ignoring the error code.

LISTING 9.1 One Way to Deal with a Problem Situation

```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();

        //Dealing with an unusual event without Java's exception
        //handling features:
        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.");
    }
}
```

Check the condition using the return value

Another Approach : Exception concept

- **Exception Concept**
  - When an error occurs (that represents an exceptional condition),
  - Exceptions cause the current program flow to be interrupted and transferred to a registered exception handling block.

- Exception handling involves a well-structured goto

9.1 Basic Exception Handling

Exception

- Java provides a way to handle certain kinds of special conditions in your program
- You can divide your codes into:
  - Sections for the normal case
  - Sections for the exceptional case
- Exception as an object
  - *Throwing* an exception
    - Signals an occurrence of unusual event during program execution
  - *Catching* the exception
    - It detects and deals with the exception at a separate section

Exception

- Throwing an exception: (호출자에게 보고)
  - Do not specify some action
  - Only creates an object that has a message.
  - Can make a constructor and throws this exception object
- Catching exceptions: (에러 발생시 처리)
  - Basic mechanism for handling exceptions

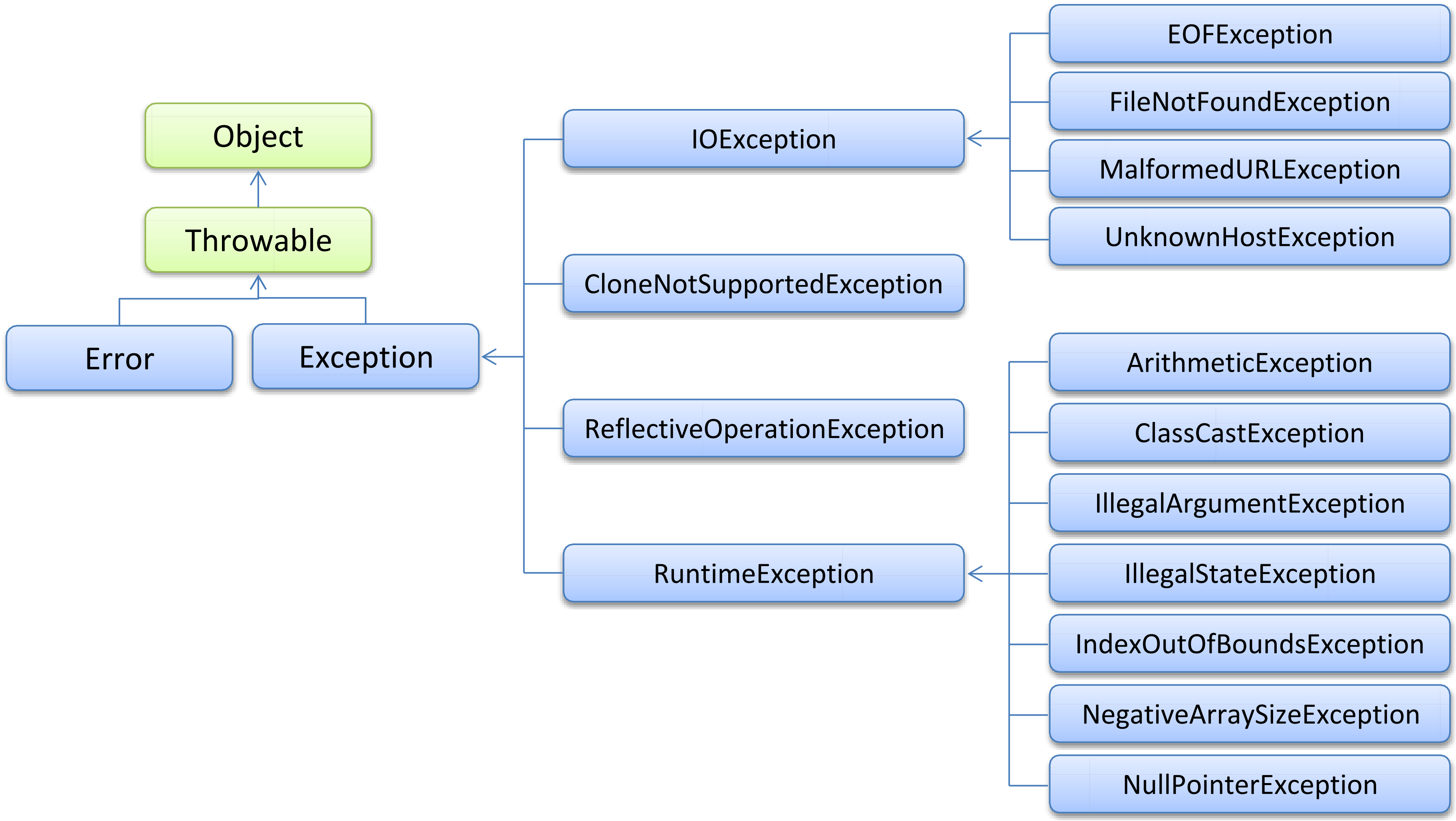
```
throw new Exception("message");
---------------------------------------------
Exception e = new
Exception(“message”);
throw e;
```

```
try {
   statements
} catch (exceptionType1 identifier1) {
   handler code for exceptionType1
} catch (exceptionType2 identifier1) {
   handler code for exceptionType2
}
```

Pre-defined exception classes

- Java has defined exception classes in Java Class Library
  - Can place method invocation in try block
  - Follow with a catch block for this type of exception
- Examples:
  - BadStringOperationException
  - ClassNotFoundException
  - IOException
  - NoSuchMethodException

Exception hierarchy (partial)



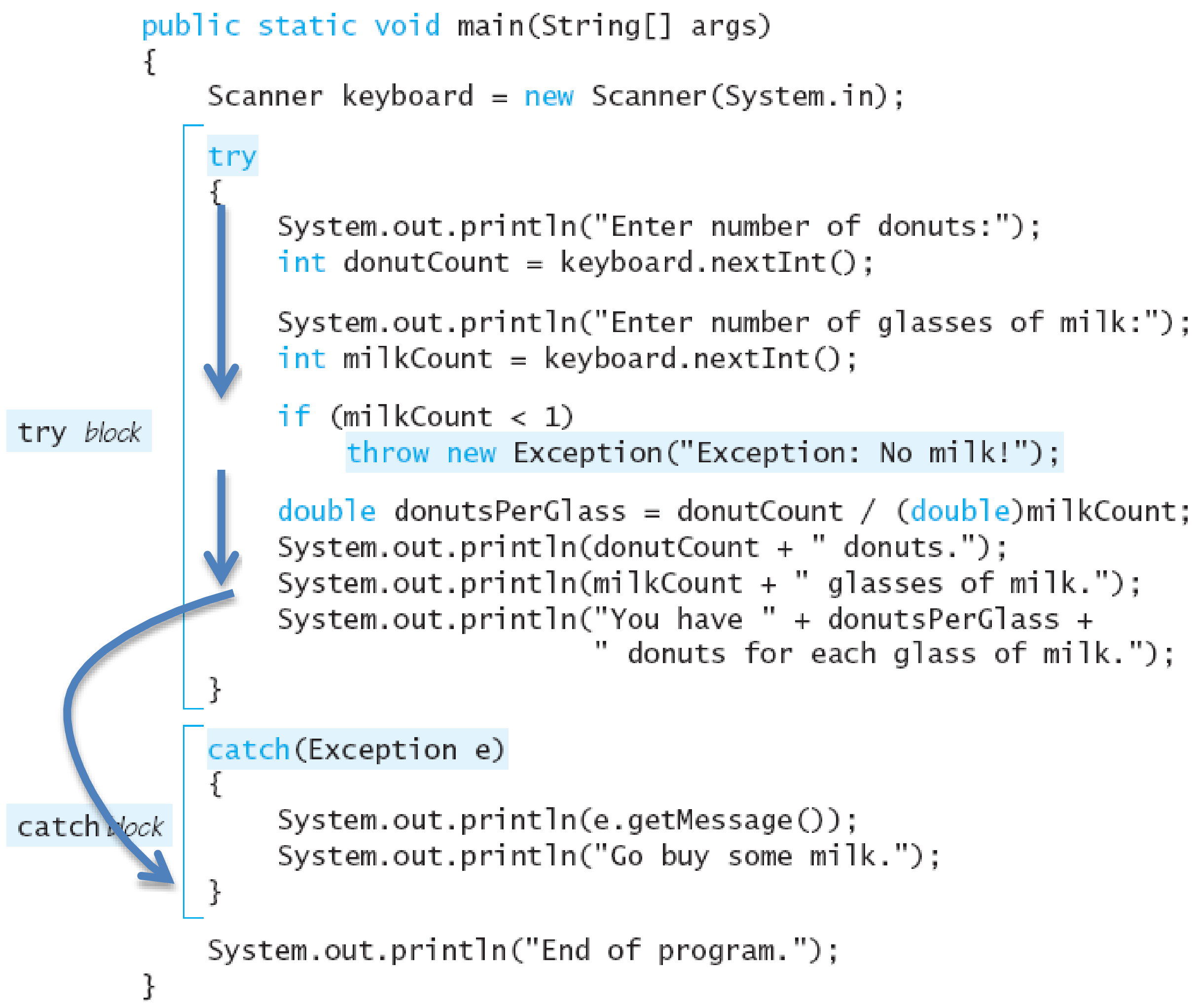
Example : Throwing

```
String readData(Scanner in) throws EOFException {
        ...
        while( ...) {
              if (!in.hasNext()) //EndOfFile encountered {
                 if(n < len)
                   throw (new Exception(“Exception: EOF Error”));
              }
              . . .
        }
        return s; }
     }
```

try/catch block

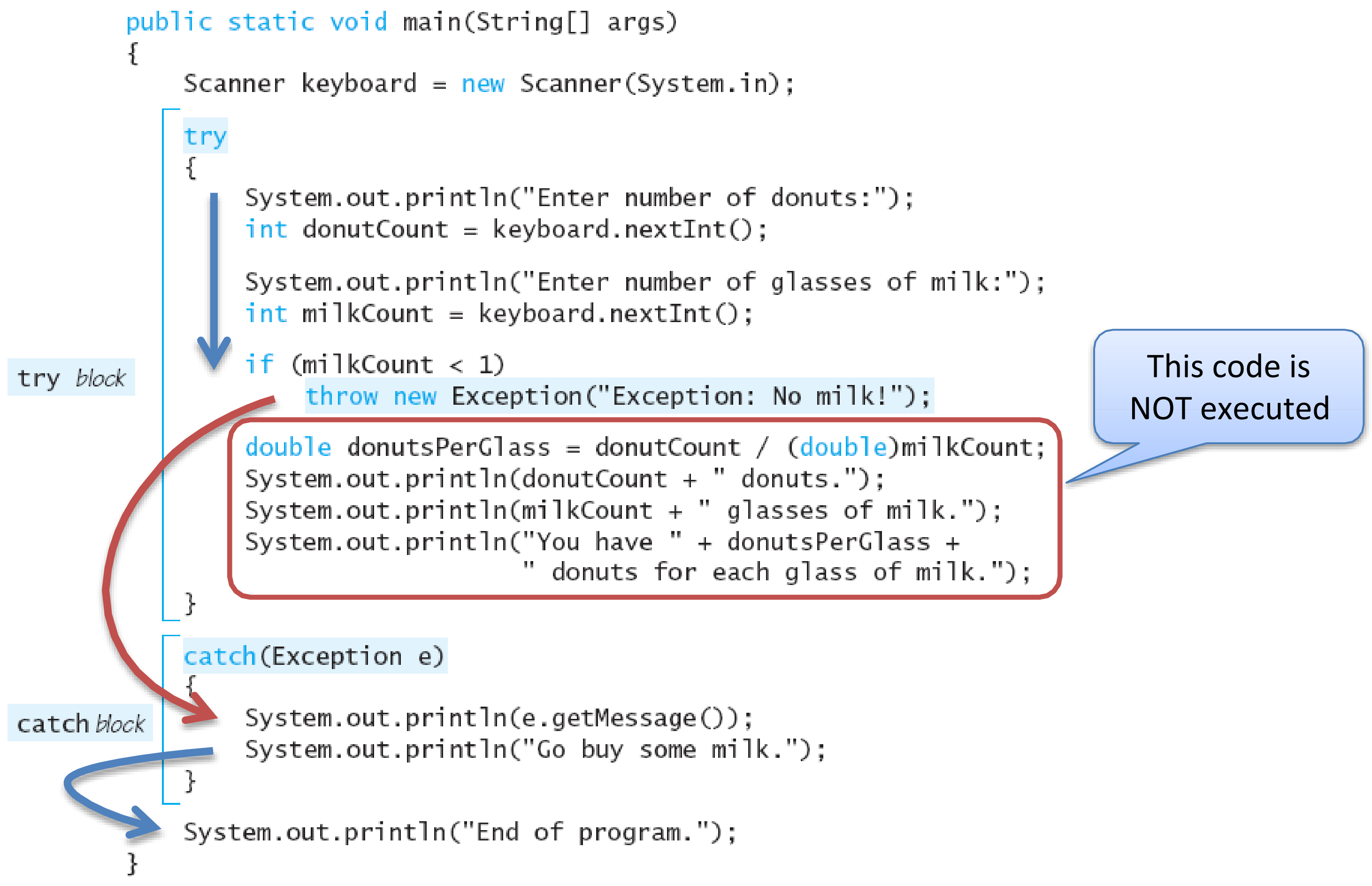
- *try* block
  - Contains code where something could possibly go wrong
  - If it does go wrong, it throws an exception
- *catch* block
  - When an exception is thrown, catch block begins execution
  - Similar to a method with a parameter
  - Parameter is the thrown exception object
  - Any number of catch blocks for different exceptions

Example: normal case



```
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.");
}
```

Example: exception case



```java
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.");
}
```

Lab: calculator

```
class Calculator{
    int left, right;
    public void setOprands(int left, int right){
        this.left = left;
        this.right = right;
    }
    public void divide(){
        System.out.print("계산결과는 ");
        System.out.print(this.left/this.right);
        System.out.print(" 입니다.");
    }
}
public class CalculatorDemo {
    public static void main(String[] args) {
      Calculator c1 = new Calculator();
            c1.setOprands(10, 0);
            c1.divide();
    }
}
```

Lab: calculator

```
Exception in thread "main" 계산결과는 java.lang.ArithmeticException: / by zero
at Calculator.divide(CalculatorDemo.java:9)
at CalculatorDemo.main(CalculatorDemo.java:17)
```

에러가 발생한 원인

에러가 발생한 함수내 위치

함수를 콜해서 에러난 지점

```
1:  class Calculator{
2:      int left, right;
3:      public void setOprands(int left, int right){
4:          this.left = left;
5:          this.right = right;
6:      }
7:      public void divide(){
8:          System.out.print("계산결과는 ");
9:          System.out.print(this.left/this.right);
10:         System.out.print(" 입니다.");
11:     }
12: }
13: public class CalculatorDemo {
14:     public static void main(String[] args) {
15:        Calculator c1 = new Calculator();
16:              c1.setOprands(10, 0);
17:              c1.divide();
18:     }
19: }
```

Lab: calculator – modify it!

```
class Calculator{
    int left, right;
    public void setOprands(int left, int right){
        this.left = left;
        this.right = right;
    }
    public void divide(){
        try {
            System.out.print("계산결과는 ");
            System.out.print(this.left/this.right);
            System.out.print(" 입니다.");
        } catch(Exception e){
            System.out.println("오류 : "+e.getMessage());
        }
    } }
public class CalculatorDemo {
    public static void main(String[] args) {
        Calculator c1 = new Calculator();
        c1.setOprands(10, 0);
        c1.divide();
        Calculator c2 = new Calculator();
        c2.setOprands(10, 5);
        c2.divide();
    }
}
```

결과 확인!!
오류가 발생하였으나 프로그램 동작에는 문제 없음

```
계산결과는 오류 : / by zero
계산결과는 2 입니다.
```

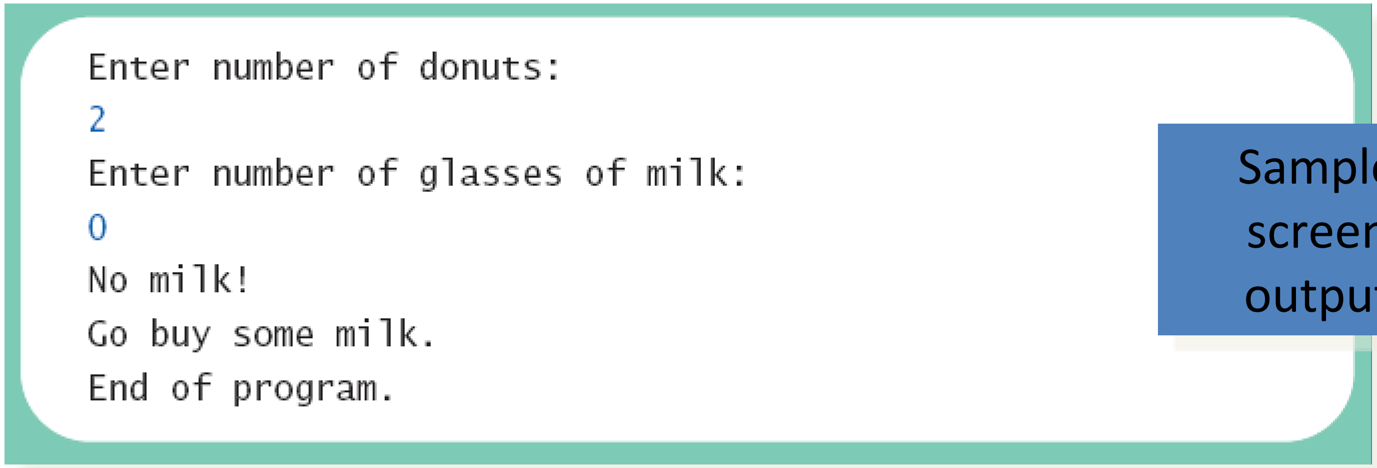
Lab: calculator –exception methods

```
class Calculator{
    int left, right;
    public void setOprands(int left, int right){
        this.left = left;
        this.right = right;
    }
    public void divide(){
        try {
            System.out.print("계산결과는 ");
            System.out.print(this.left/this.right);
            System.out.print(" 입니다.");
        } catch(Exception e){
            System.out.println("\n 오류 출력 1 \n "+e.getMessage());
            System.out.println("\n 오류 출력 2 \n "+e.toString());
            System.out.println("\n 오류 출력 3");
            e.printStackTrace();
        }
    }
}
public class CalculatorDemo {
    public static void main(String[] args) {
        Calculator c1 = new Calculator();
        c1.setOprands(10, 0);
        c1.divide();
    }
}
```

```
계산결과는
 오류 출력 1
 / by zero

 오류 출력 2
 java.lang.ArithmeticException: / by zero

 오류 출력 3
java.lang.ArithmeticException: / by zero
at Calculator.divide(CalculatorDemo.java:10)
at
CalculatorDemo.main(CalculatorDemo.java:24)
```

Exceptions in Java

- Consider a program to assure us of a sufficient supply of milk
- View possible solution, listing 9.1 & 9.2
  class GotMilk



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

Sample screen output

Class GotMilk, ExceptionDemo

```java
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();

    //Dealing with an unusual event without Java's exception
    //handling features:
    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.");
}
```

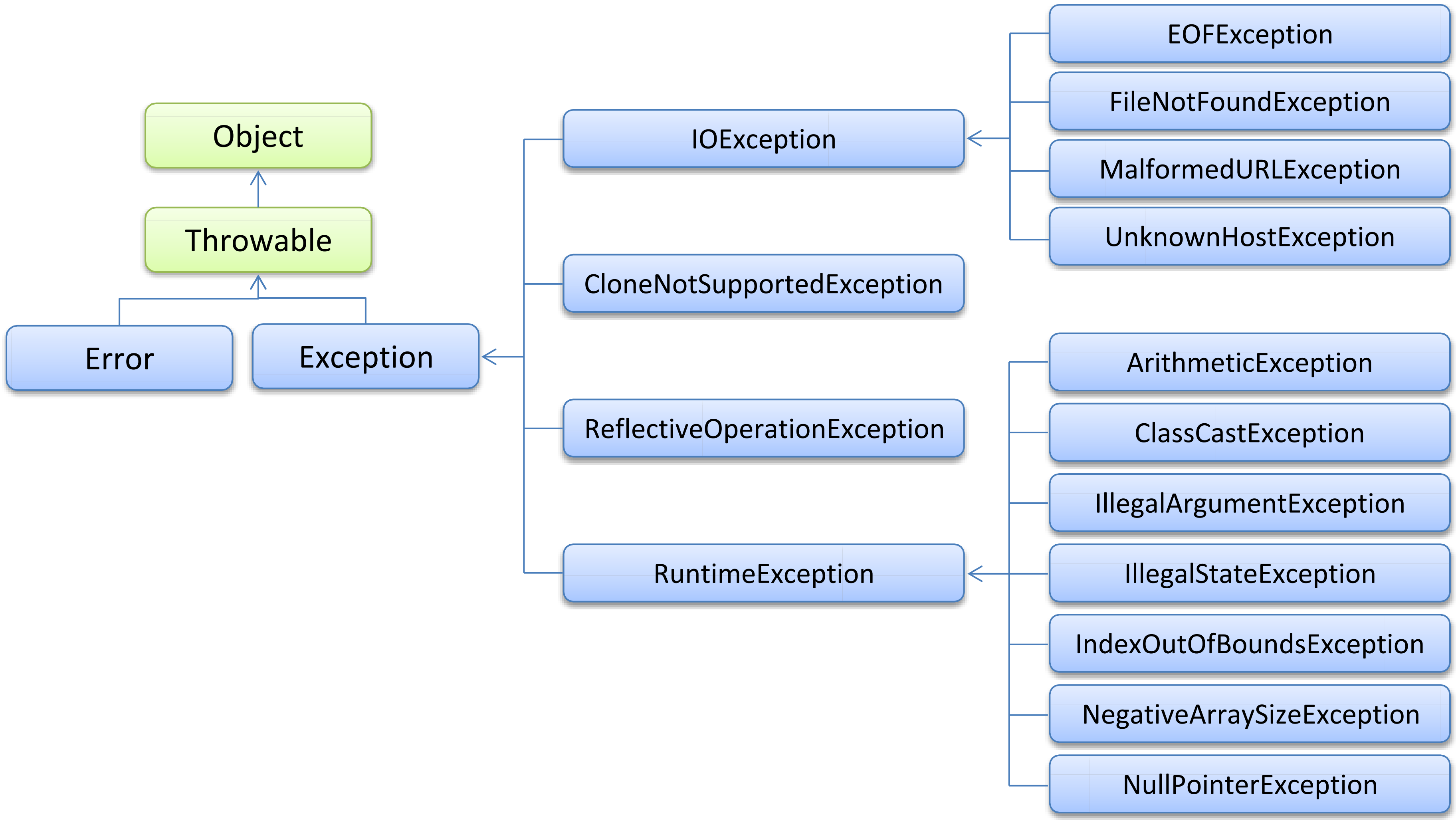
```java
Scanner keyboard = new Scanner(System.in);
try
{
    System.out.println("Enter number of donuts:");
    int donutCount = keyboard.nextInt();

    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.");
```

finally block

- Possible to add after the sequence of catch blocks
- Code in finally block executed
  - Whether or not an exception is thrown
  - Whether or not a required catch exists
- A good place to put clean-up code, i.e., close open files

```
try {
    // Code to try: possibly throws an exception
}
catch (IOException x) { ... }
catch (Exception x) { ... }
finally {
    // This code is ‘always’ executed
}
```

Other exceptions



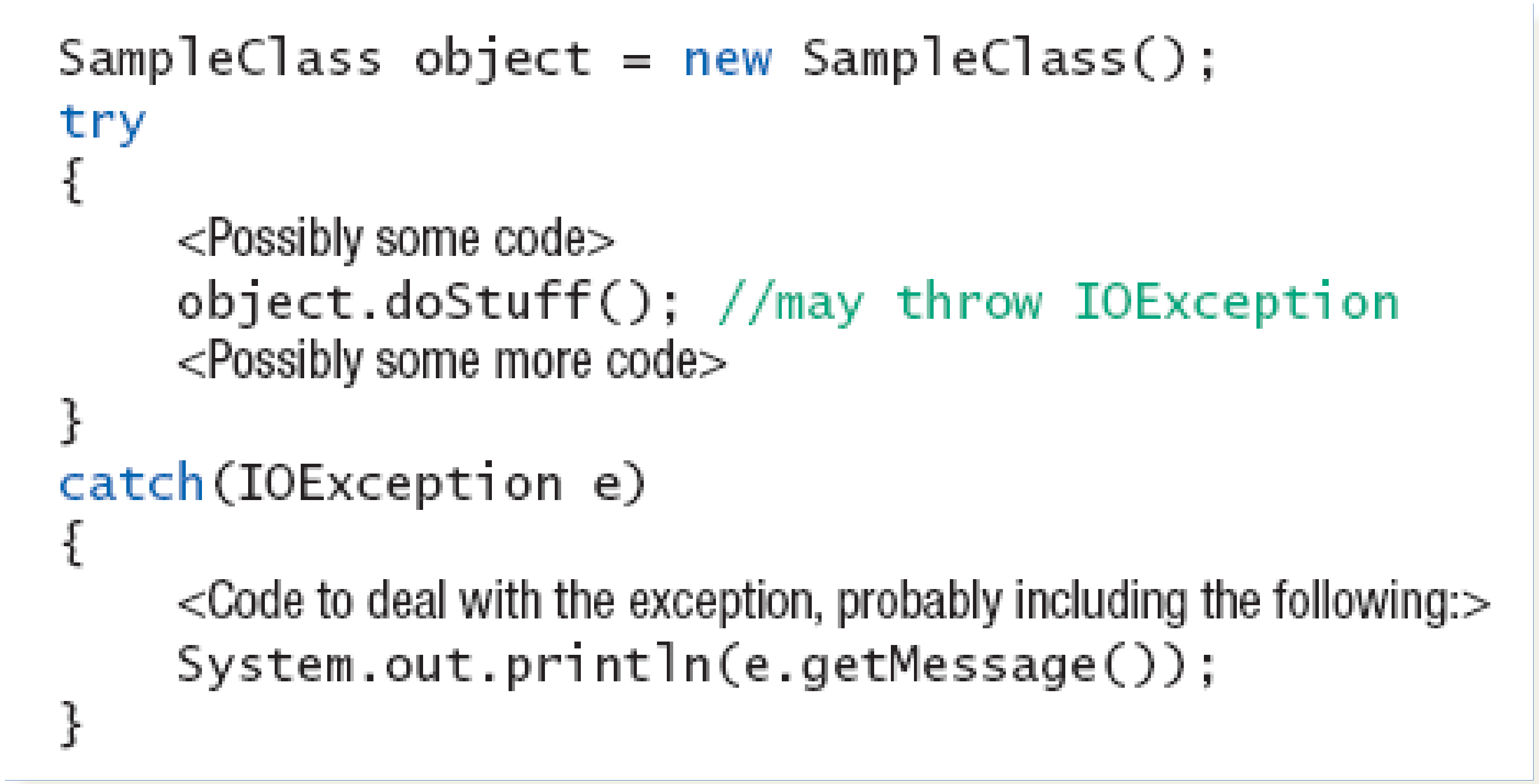
Example 1

```
public void read(String fileName) {
      try {
        InputStream in = new FileInputStream(fileName);
        int b;

       //the read() method below is one which will throw
  an IOException
        while ((b = in.read()) != -1) {
          process input
        }
      } catch (IOException e) {
        e.printStackTrace();
      }
}
```

Example 2

```
      try {
                access the database …
       } catch (SQLException e) {
         Throwable se = new ServletException(“database   error”);
         se.setCause(e);
         throw se;
       }
```

Example 3



```
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());
}
```

Lab: multiple exceptions

```
class ArrayData{
    private int[] arr = new int[3];
    ArrayData(){
        arr[0]=0;
        arr[1]=10;
        arr[2]=20;
    }
    public void z(int first, int second){
        System.out.println(arr[first] / arr[second]);
    }
}
public class ExceptionDemo {

    public static void main(String[] args) {
        ArrayData a = new ArrayData();
        a.z(10, 2); //test1
        a.z(1, 0);  //test2
    }
}
```

ctrl + space 키로 컴파일러 추천 코드 확인 가능

Modify it using multiple exception class!

```
class ArrayData{
    private int[] arr = new int[3];
    ArrayData(){
        arr[0]=0;
        arr[1]=10;
        arr[2]=20;
    }
    public void z(int first, int second){
    try {
            System.out.println(arr[first] / arr[second]);
        } catch(ArrayIndexOutOfBoundsException e){
            System.out.println("ArrayIndexOutOfBoundsException");
        } catch(ArithmeticException e){
            System.out.println("ArithmeticException");
        } catch(Exception e){
            System.out.println("Exception");
        }
    }
}
public class ExceptionDemo {
    public static void main(String[] args) {
    ArrayData a = new ArrayData();
        a.z(10, 2); //test1
        a.z(1, 0);  //test2
    }
}
```

ctrl + space 키로 컴파일러 추천 코드 확인 가능

9.2 Defining Your Own Exception Classes (Optional topic)

Defining Your Own Exception Classes

- You can define your own exception classes
- Must be derived class of some predefined exception class
  - Any exception class can be derived
  - Textbook uses classes derived from class Exception
- Method getMessage defined in exception classes
  - Returns string passed as argument to constructor
  - If no actual parameter used, default message returned

```
catch(Exception e) {
    System.out.println(e.getMessage());
    }
```

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
  - Don’t override inherited *getMessage()* in exception classes

Example

- List 9.5

```
public class DivideByZeroException extends Exception
{
    public DivideByZeroException ()
    {
        super ("Dividing by Zero!");
    }

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

Lab: DivideByZeroDemo

- Different runs of the program

```
Enter numerator:
5
Enter denominator:
10
5/10 = 0.5
End of program.
```

Sample screen output 1

```
Enter numerator:
5
Enter denominator:
0
Dividing by Zero!
Try again.
Enter numerator:
5
Enter denominator:
Be sure the denominator is not zero.
10
5/10 = 0.5
End of program.
```

Sample screen output 2

```
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.
```

Sample screen output 3

Lab: modify the code

```
import java.util.Scanner;
public class DivideByZeroDemo {
    private int numerator;
    private int denominator;
    private double quotient;
    public static void main (String [] args)    {
        DivideByZeroDemo oneTime = new DivideByZeroDemo ();
        oneTime.doIt ();
    }
    public void doIt ()    {
        System.out.println ("Enter numerator:");
         Scanner keyboard = new Scanner (System.in);
         numerator = keyboard.nextInt ();
         System.out.println ("Enter denominator:");
         denominator = keyboard.nextInt ();
        quotient = numerator / (double) denominator;
         System.out.println (numerator + "/" + denominator +  " = "
         + quotient);
        System.out.println ("End of program.");
    }

// code continues to the next page
```

Make the code robust !

Lab: modify the code

```
    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);
    }
}
```

9.3 More About Exception Classes

Declaring Exceptions

- You caught the exception inside the method in Section 9.2
  - and, your code handled the exception immediately
  - **Problem ?**

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

```
public void sampleMethod() throws DivideByZeroException
```

  - Pass the responsibility ("pass the buck") to handle exception from the method itself to any method that calls it

Declaring Exceptions

```
public void methodA() throws IOException
```

Hey, methodB(),
If you invoke me, you must worry about any `IOException` that I throw

```
public void methodB() {
  try
  {
    obj.methodA();
  }
  catch(IOException e){
        System.out.println(e.getMessage());
      System.exit(0);
  }
}
```

OK!!

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

Lab.: class DoDivision

- If a method throws exception and exception not caught inside the method
  - Method ends immediately after exception thrown

- Recall class `DivideByZeroDemo` for lab

Recall class DivideByZeroDemo

- List 9.5

```
public class DivideByZeroException extends Exception
{
    public DivideByZeroException ()
    {
        super ("Dividing by Zero!");
    }

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

```
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 ();

        doIt.doNormalCase ();

        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);
    }
    // giveSecondChance() is given in 9.6
}
```

Modify here!
if the denominator is zero,
give the user second chance by
invoking giveSecondChance();

doNormalCase 에서 받은
exception을 main에서 처리

Differences? (again)

```
import java.util.Scanner;
public class DoDivision {
    …………
    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);
    }
}
```

VS

```
public class DivideByZeroDemo {
    …..
    public static void main (String [] args)    {
        DivideByZeroDemo oneTime = new DivideByZeroDemo ();
        oneTime.doIt ();
    }
    public void doIt ()    {
    try{      …
       System.out.println ("Enter denominator:");
       denominator = keyboard.nextInt ();
       if (denominator == 0)          throw new DivideByZeroException ();
       quotient = numerator / (double) denominator;
       System.out.println (numerator + "/" + denominator +  " = " + quotient);
      }
    catch(DivideByZeroException e){
       System.out.println(e.getMessage());
       giveSecondChance();
       }
      System.out.println ("End of program.");
    }
```

Differences between throw vs throws

- Keyword **`throw`** used to throw exception
  - Catch the possible exception in a catch block within the method definition
- Keyword **`throws`** used in method heading to declare an exception
  - Declare the possible exception by writing a throws clause in the method’s heading
  - Let whoever uses the method worry about how to handle the exception

Kinds of Exceptions

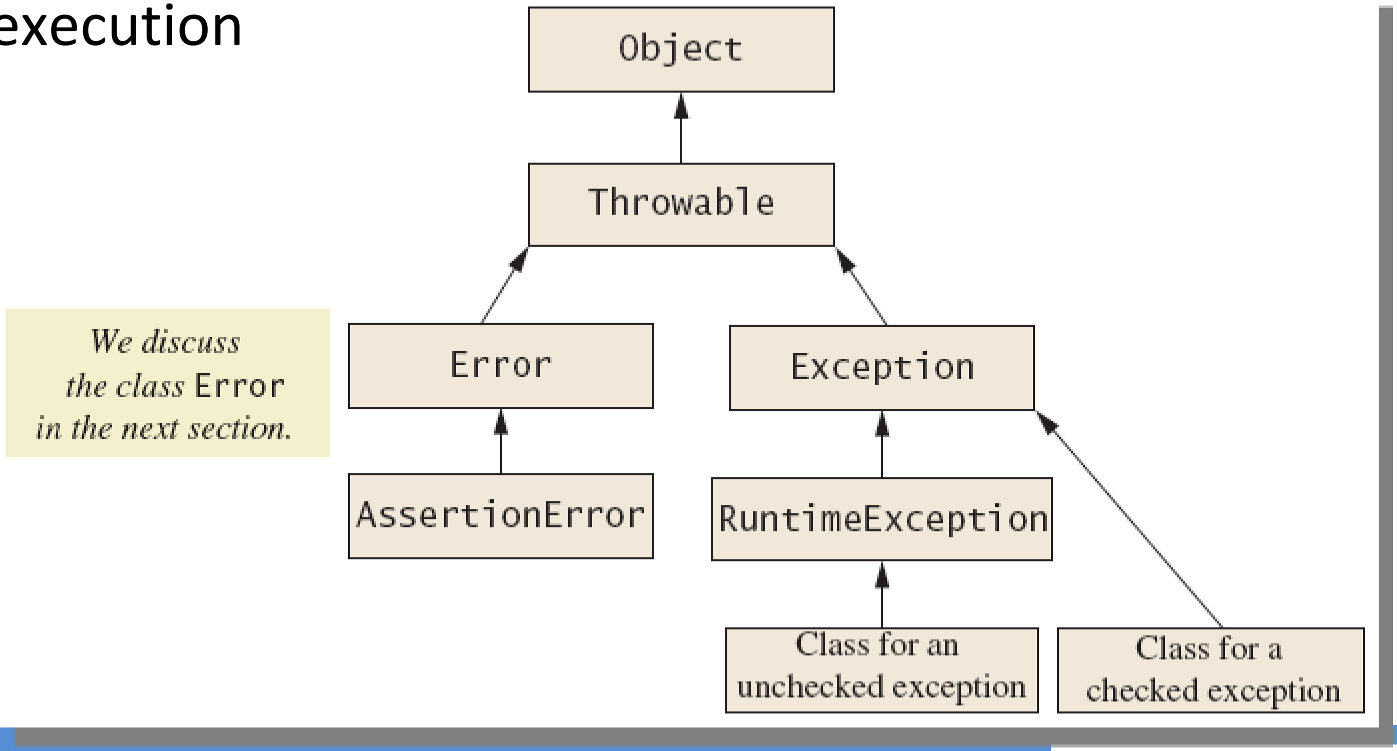
- In most cases, exception is caught …
  - In a catch block … or
  - Be declared in throws clause
- But Java has exceptions you don’t need to account for
- Categories of exceptions
  - Checked exceptions
  - Unchecked exceptions

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 coding problems exist, **should be fixed**

Kinds of Exceptions

- Examples why unchecked exceptions to are thrown
  - Use array index out of bounds or division by zero
  - Uncaught runtime exception terminates program execution


Object
Throwable
Error
Exception
AssertionError
RuntimeException
Class for an unchecked exception
Class for a checked exception
We discuss the class Error in the next section.

Errors

- An *error* is an object of class `Error`
  - Similar to an unchecked exception
  - Need not catch or declare in throws clause
  - Object of class `Error` generated when abnormal conditions occur
- Errors are more or less beyond your control
  - Require change of program to resolve

Multiple Throws and Catches

- Which one is better ?

A

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

B

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

//The second catch *block can never be reached.*

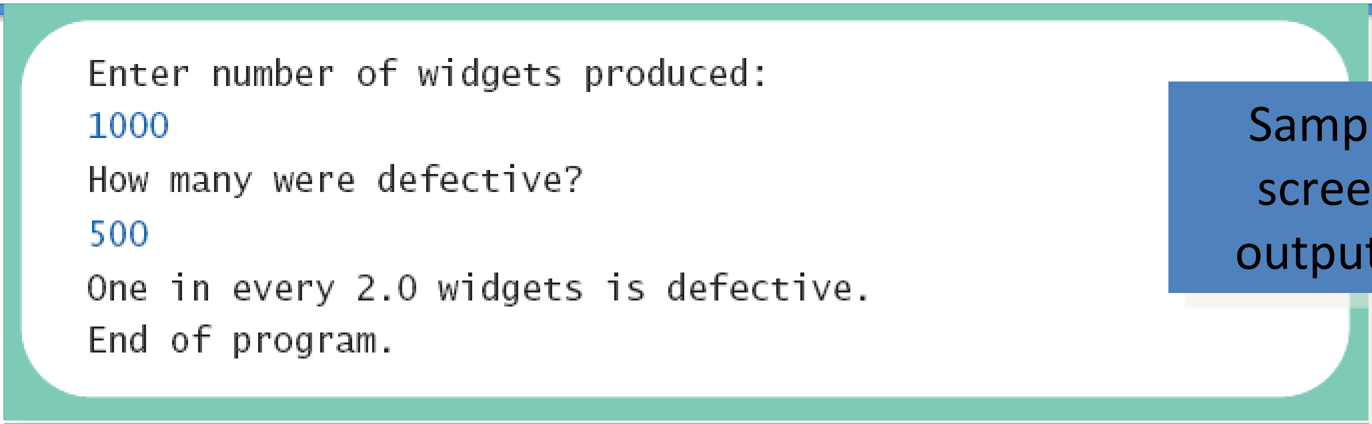
Multiple Throws and Catches

- 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 example program, listing 9.8
  class TwoCatchesDemo
- View exception class used, listing 9.9
  class NegativeNumberException

Multiple Throws and Catches

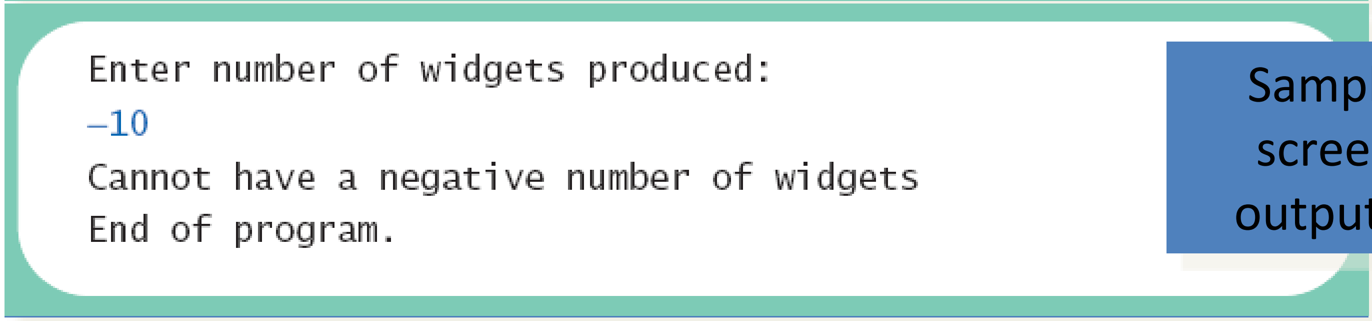
- When catching multiple exceptions, the order of the catch blocks can be important.
- When an exception is thrown in a try block, the catch blocks are examined in order of appearance.

Lab: Multiple Throws and Catches



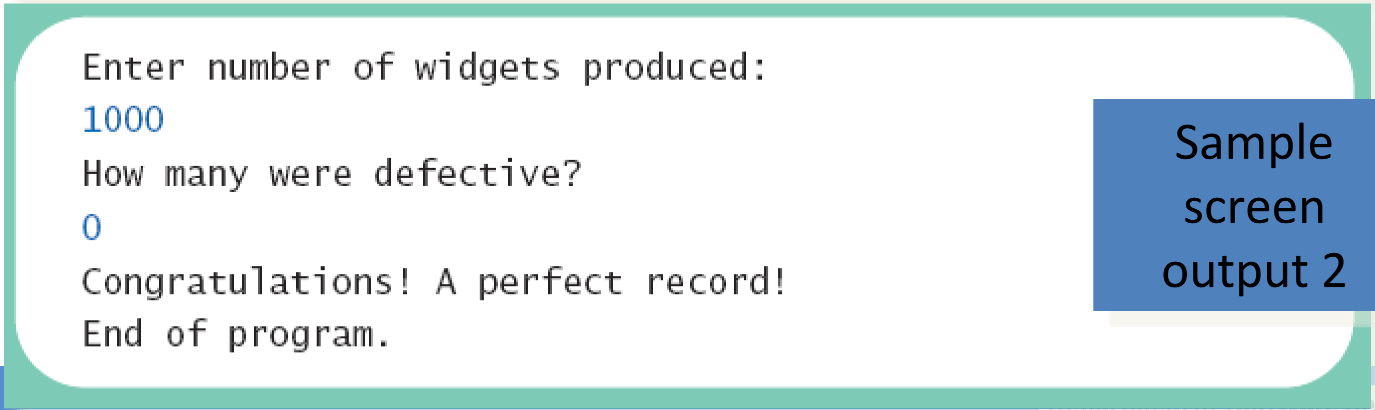
```
Enter number of widgets produced:
1000
How many were defective?
500
One in every 2.0 widgets is defective.
End of program.
```

Sample screen output 1



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

Sample screen output 2



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

Sample screen output 2

```
    public static void main (String [] args)
    {
        try
        {
            System.out.println ("Enter number of widgets produced:");
            Scanner keyboard = new Scanner (System.in);
            int widgets = keyboard.nextInt ();
            if (widgets < 0)
                throw new NegativeNumberException ("widgets");
            System.out.println ("How many were defective?");
            int defective = keyboard.nextInt ();
            if (defective < 0)
                throw new NegativeNumberException ("defective
widgets");
            double ratio = exceptionalDivision (widgets, defective);
            System.out.println ("One in every " + ratio +
                    " widgets is defective.");
        }



        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;
    }
```

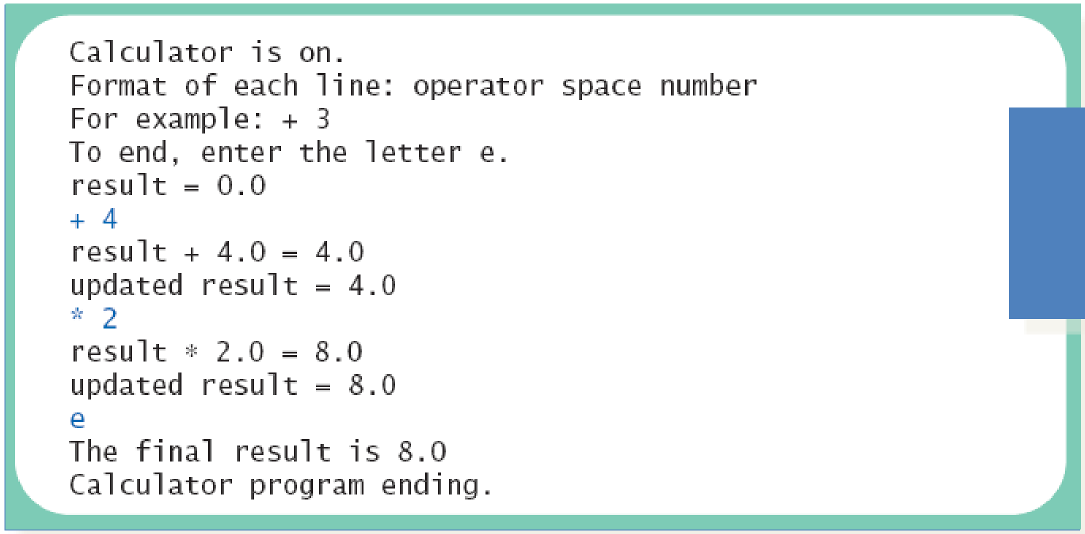
```
catch (DivideByZeroException e)
{
   …
}
catch (NegativeNumberException e)
{
   …
}
```

Lab: case study calculator2

- A Line-Oriented Calculator
  - Should do addition, subtraction, division, multiplication
  - Will use line input/output
- User will enter
  - Operation, space, number
  - Calculator displays result

Lab: calculator2

- View exception class, listing 9.10
  class UnknownOpException
- View first version of calculator, listing 9.11
  class PreLimCalculator



```
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
result * 2.0 = 8.0
updated result = 8.0
e
The final result is 8.0
Calculator program ending.
```

Sample screen output

Lab: calculator2

- Proposed initial methods
  - Method to reset value of result to zero
  - Method to evaluate result of one operation
  - Method doCalculation to perform series of operations
  - Accessor method getResult: returns value of instance variable result
  - Mutator method setResults: sets value of instance variable result

Lab: calculator2

```
import java.util.Scanner;
public class PrelimCalculator{
    private double result;
    private double precision = 0.0001; // Numbers this close 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 (e.g) + 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);
           }
       }
   }
   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;
   }
}
```

Lab: calculator2

```
public class DivideByZeroException extends Exception{
    public DivideByZeroException( )    {
        super("Dividing by Zero!");
    }
    public DivideByZeroException(String message)    {
        super(message);
    }
}

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);
    }
}
```

Lab: Modify calculator2!

- Final version adds exception handling
- Ways to handle unknown operator
  - Catch exception in method evaluate
  - Let evaluate throw exception, catch exception in doCalculation
  - Let evaluate, doCalculation both throw exception, catch in main

Change main

```
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.");
   }
```

Add handling methods

```
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);
       }
   }
```