CSY2030
Systems Design & Development
Exception Handling

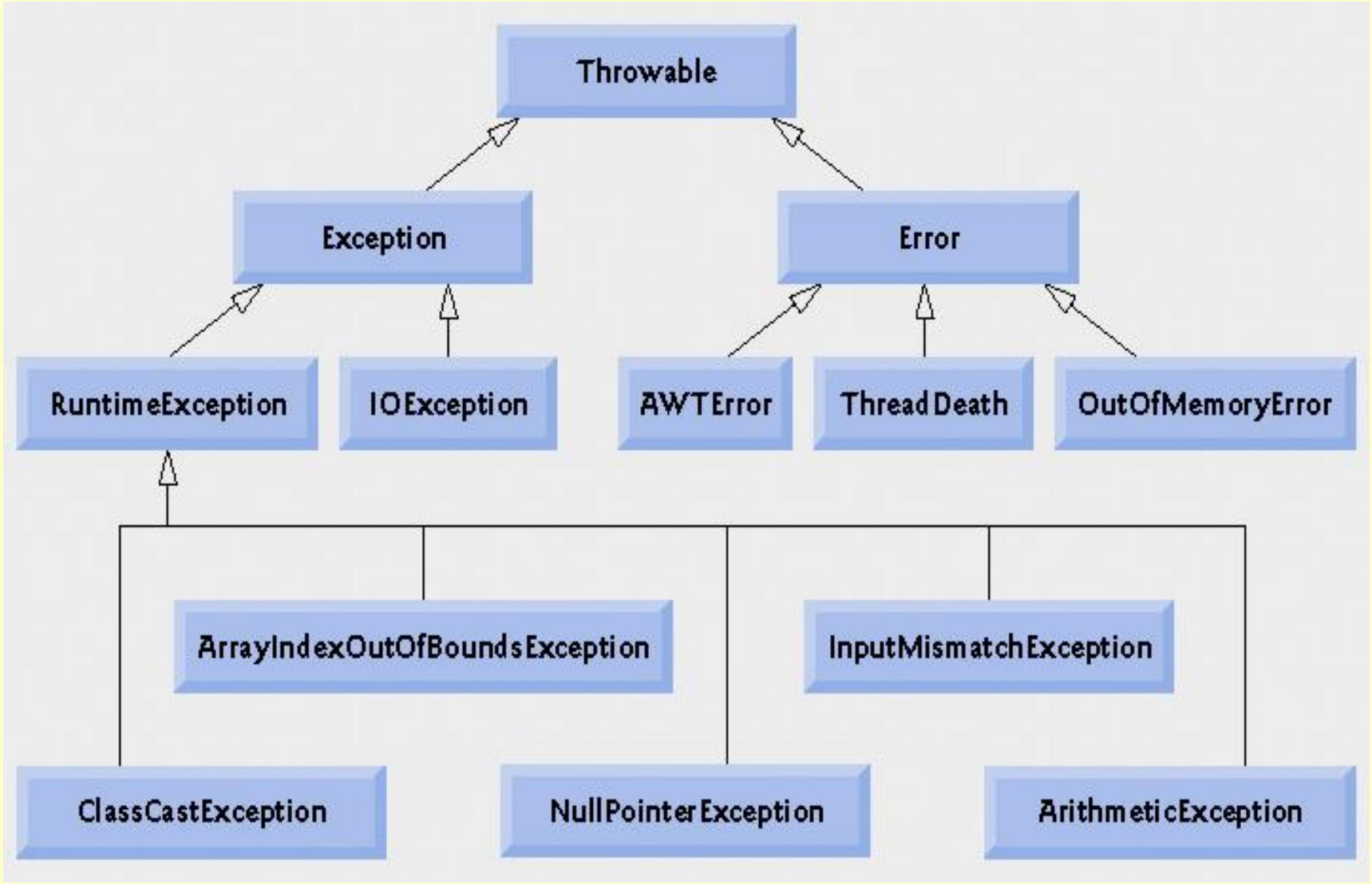
Exceptions

- Exceptions are used to handle bad things such as the following:
  - I/O errors e.g file doesn’t exist
  - *runtime* errors e.g division by zero
  - when a function fails to fulfill its specification
- Without exceptions the program will crash
- With exceptions you can restore program stability (or exit gracefully)

Exceptions

- Examples of Exceptions include the following:
  - **ArrayIndexOutOfBoundsException** – an attempt is made to access an element past the end of an array
  - **ClassCastException** – an attempt is made to cast an object that does not have an *is-a* relationship with the type specified in the cast operator
  - **NullPointerException** – when a null reference is used where an object is expected
  - **ArithmeticException** – e.g used for division by 0

Exceptions Hierarchy



Exception Handling Terms

- **throw** – used to generate an exception or to describe an instance of an exception
- **try** – used to enclose a segment of code that may throw an exception
- **catch** – placed directly after the try block to handle one or more exception types
  - Can have multiple catches if try block can throw multiple exceptions
- **finally** – optional statement used after a try-catch block to run a segment of code regardless if a exception is generated

Exception-handling Syntax

```
try
{
        // <code segment that may throw an exception..>
}
catch (ExceptionType1 x)
{
         // handle ExceptionType1 error
}
…
catch (ExceptionTypeN x)
{
         // handle ExceptionTypeN error
}
finally
{
        // invariant code ("always" executes)
        // finally is optional code segment.
}
```

Example

- Consider the following code:

```
import java.io.FileReader;

public class Tester
{
	public int countChars(String fileName)
	{	FileReader r = new FileReader(fileName);
		int total = 0;
		while( r.ready() )
		{
			r.read();
			total++;
		}
		r.close();
		return total;
	}
}
```

Problem with above code is that the file may not exist i.e exception java.io.FileNotFoundException will be generated - this must be caught or declared to be thrown.

Handling Checked Exceptions

- In the method *countChars* on the previous slide there are in fact 4 statements that can generate checked exceptions:
  - the *FileReader* constructor
  - the *ready* method
  - the *read* method
  - the *close* method
- To deal with the exceptions we can either state this method throws an Exception of the proper type or handle the exception within the method itself

Methods that throw Exceptions

- It may be that we don't know how to deal with an error within the method that can generate it
- In this case we will pass the buck to the method that called us
- The keyword **throws** is used to indicate a method has the possibility of generating an exception of the stated type
- Now any method calling ours must also throw an exception or handle it

Using the **throws** Keyword

```
public int countChars(String fileName)
        throws FileNotFoundException, IOException
{
        int total = 0;
        FileReader r = new FileReader(fileName);
        while( r.ready() )
        {
                r.read();
                total++;
        }
        r.close();
        return total;
}
```

Now any method calling ours must also throw an exception or handle it

Using **try-catch** Blocks

- If you want to handle the a checked exception locally then use use the keywords **try** and **catch**
- the code that could cause an exception is placed in a block of code preceded by the keyword **try**
- the code that will handle the exception if it occurs is placed in a block of code preceded by the keyword **catch**

Sample **try** and **catch** Blocks

```
public int countChars(String fileName)
{
	int total = 0;
	try
	{	FileReader r = new FileReader(fileName);
		while( r.ready() )
		{
			r.read();
			total++;
		}
		r.close();
	}
	catch(FileNotFoundException e)
	{	System.out.println("File named " + fileName + "not found. " + e);
		total = -1;
	}
	catch(IOException e)
	{	System.out.println("IOException occured " +
			"while counting chars. " + e);
		total = -1;
	}
	return total;
}
```

What Happens When Exceptions Occur

- If an exception is thrown then the normal flow of control of a program halts
- Instead of executing the regular statements the Java Runtime System starts to search for a matching catch block
- The first matching catch block based on data type is executed
- When the catch block code is completed the program does not "go back" to where the exception occurred.
  - It finds the next regular statement after the catch block

Excessive try catch Block

```
public int countChars(String fileName)
{               int total = 0;
                FileReader r = null;
                try
                {     r = new FileReader(fileName);
                }
                 catch(FileNotFoundException e)
                {   System.out.println("File named " + fileName + "not found. " + e);
                                total = -1;
                }
                try
                {       while( r.ready() )
                                {   try
                                    {
                                                r.read();
                                     }
                                     catch(IOException e)
                                                {System.out.println("IOException " + "occurred while counting “ + "chars. " + e);
                                                  total = -1;
                                                }
                                                total++;
                                }
                }
                catch(IOException e)
                {     System.out.println("IOException occurred while counting chars. " + e);
                     total = -1;                }
                try
                {     r.close();
                }
                catch(IOException e)
                {     System.out.println("IOException occurred while counting chars. " + e);
                     total = -1;
                 }
                return total;
}
```

Throwing Exceptions Yourself

- If you wish to throw an exception in your code you use the throw keyword
- Most common would be for an unmet precondition

```
public class Circle
{
        private int iMyRadius;
        /** pre: radius > 0
         */
        public Circle(int radius)
        {       if (radius <= 0)
                        throw new IllegalArgumentException
                                ("radius must be > 0. "
                                + "Value of radius: " + radius);
                iMyRadius = radius;
        }
}
```

Another Example

```
1  // Fig. 13.1: DivideByZeroNoExceptionHandling.java
2  // An application that attempts to divide by zero.
3  import java.util.Scanner;
4
5  public class DivideByZeroNoExceptionHandling
6  {
7     // demonstrates throwing an exception when a divide-by-zero occurs
8     public static int quotient( int numerator, int denominator )
9     {
10       return numerator / denominator; // possible division by zero
11    } // end method quotient
12
13    public static void main( String args[] )
14    {
15       Scanner scanner = new Scanner( System.in ); // scanner for input
16
17       System.out.print( "Please enter an integer numerator: " );
18       int numerator = scanner.nextInt();
19       System.out.print( "Please enter an integer denominator: " );
20       int denominator = scanner.nextInt();
21
22       int result = quotient( numerator, denominator );
23       System.out.printf(
24          "\nResult: %d / %d = %d\n", numerator, denominator, result );
25    } // end main
26 } // end class DivideByZeroNoExceptionHandling
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at
DivideByZeroNoExceptionHandling.quotient(DivideByZeroNoExceptionHandling.java:10)
        at
DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:22)
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at
DivideByZeroNoExceptionHandling.main(DivideByZeroNoExceptionHandling.java:20)
```

```
1  // Fig. 13.2: DivideByZeroWithExceptionHandling.java
2  // An exception-handling example that checks for divide-by-zero.
3  import java.util.InputMismatchException;
4  import java.util.Scanner;
5
6  public class DivideByZeroWithExceptionHandling
7  {
8     // demonstrates throwing an exception when a divide-by-zero occurs
9     public static int quotient( int numerator, int denominator )
10       throws ArithmeticException
11    {
12       return numerator / denominator; // possible division by zero
13    } // end method quotient
14
15    public static void main( String args[] )
16    {
17       Scanner scanner = new Scanner( System.in ); // scanner for input
18       boolean continueLoop = true; // determines if more input is needed
19
20       do
21       {
22          try // read two numbers and calculate quotient
23          {
24             System.out.print( "Please enter an integer numerator: " );
25             int numerator = scanner.nextInt();
26             System.out.print( "Please enter an integer denominator: " );
27             int denominator = scanner.nextInt();
28
```

`throws` clause specifies that method `quotient` may throw an `ArithmeticException`

Repetition statement loops until `try` block completes successfully

`try` block attempts to read input and perform division

Retrieve input; `InputMismatchException` thrown if input not valid integers

```
29             int result = quotient( numerator, denominator );
30             System.out.printf( "\nResult: %d / %d = %d\n", numerator,
31                denominator, result );
32             continueLoop = false; // input successful; end looping
33          } // end try
34          catch ( InputMismatchException inputMismatchException )
35          {
36             System.err.printf( "\nException: %s\n",
37                inputMismatchException );
38             scanner.nextLine(); // discard input so user can try again
39             System.out.println(
40                "You must enter integers. Please try again.\n" );
41          } // end catch
42          catch ( ArithmeticException arithmeticException )
43          {
44             System.err.printf( "\nException: %s\n", arithmeticException );
45             System.out.println(
46                "Zero is an invalid denominator. Please try again.\n" );
47          } // end catch
48       } while ( continueLoop ); // end do...while
49    } // end main
50 } // end class DivideByZeroWithExceptionHandling
```

If we have reached this point, input was valid and `denominator` was non-zero, so looping can stop

Call method `quotient`, which may throw `ArithmeticException`

Catching `InputMismatchException` (user has entered non-integer input)

Notify user of error made

Catching `ArithmeticException` (user has entered zero for denominator)

If line 32 was never successfully reached, loop continues and user can try again

```
Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: 0

Exception: java.lang.ArithmeticException: / by zero
Zero is an invalid denominator. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```

```
Please enter an integer numerator: 100
Please enter an integer denominator: hello

Exception: java.util.InputMismatchException
You must enter integers. Please try again.

Please enter an integer numerator: 100
Please enter an integer denominator: 7

Result: 100 / 7 = 14
```