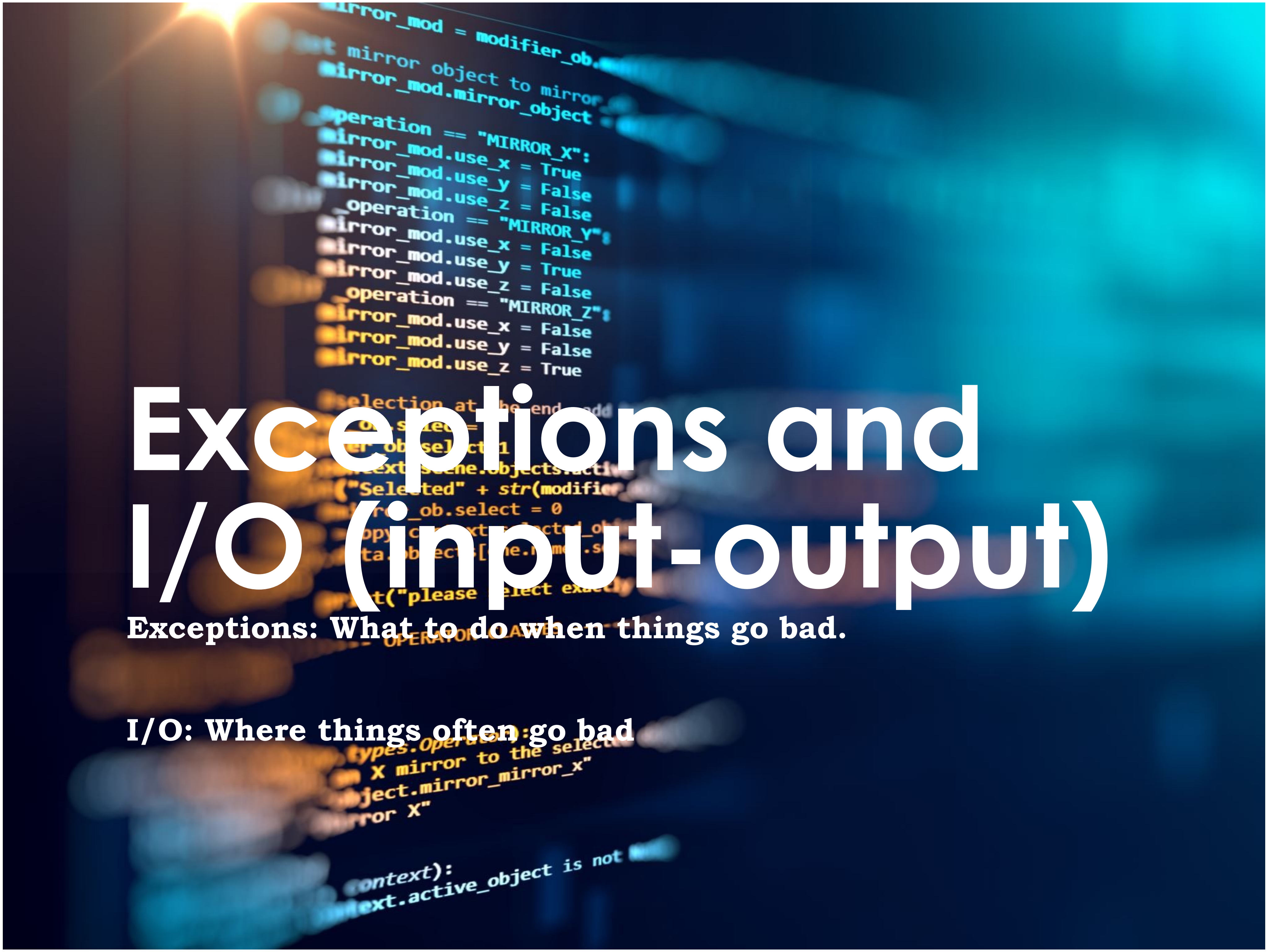
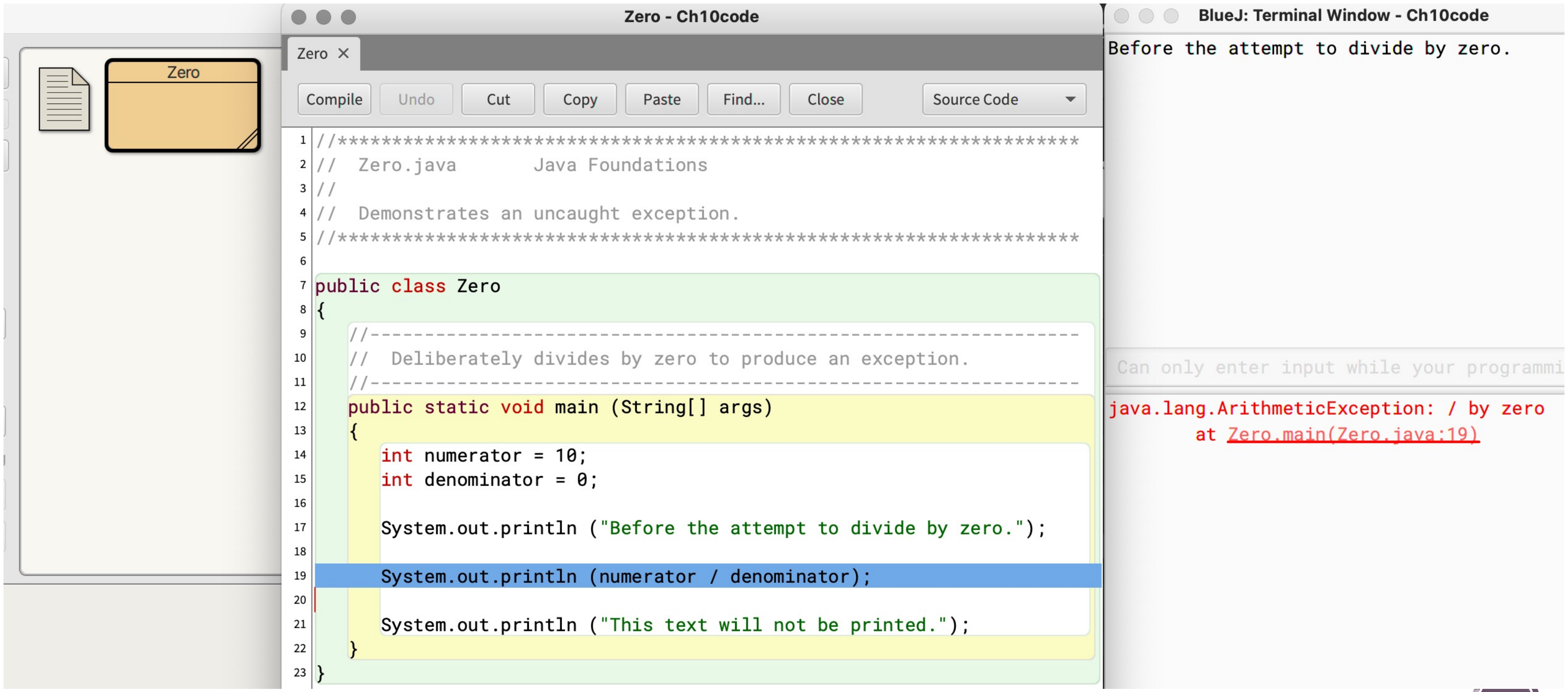


Exceptions and I/O (input-output)

Exceptions: What to do when things go bad.

I/O: Where things often go bad

Dividing by zero is bad!



```java
//********************************************************************
//  Zero.java       Java Foundations
//
//  Demonstrates an uncaught exception.
//********************************************************************

public class Zero
{
   //-----------------------------------------------------------------
   //  Deliberately divides by zero to produce an exception.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int numerator = 10;
      int denominator = 0;

      System.out.println ("Before the attempt to divide by zero.");

      System.out.println (numerator / denominator);

      System.out.println ("This text will not be printed.");
   }
}
```

```
Before the attempt to divide by zero.
```

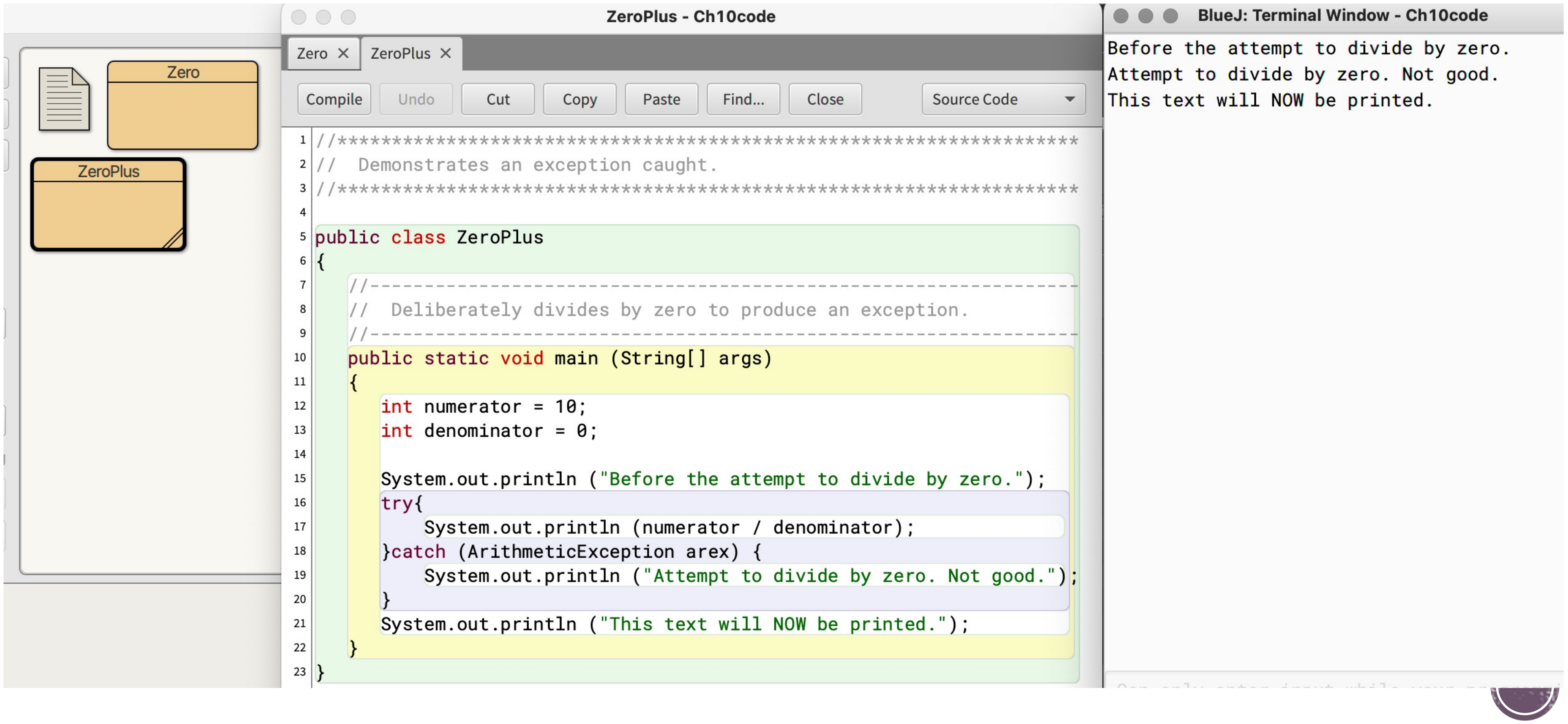
```
Can only enter input while your programmi
```

```
java.lang.ArithmeticException: / by zero
	at Zero.main(Zero.java:19)
```

10.1 – Exceptions vs Errors

- You have been coding for a while and you may have encountered some exceptions. Here are some of them:
  - Division by 0 in computing expression
  - Array index out of bounds
  - Null pointer cannot be followed
  - Generic I/O problems (e.g., no space on disk to save file, file not found, etc)
  - No permissions to save a file on the disk
- An ***exception*** is an object describing unusual or erroneous situation
- (An ***error*** is also an object, but it represents a **unrecoverable** situation and should **not** be caught)

Dividing by zero recovery!



```java
//*********************************************************************
//  Demonstrates an exception caught.
//*********************************************************************

public class ZeroPlus
{
    //-----------------------------------------------------------------
    //  Deliberately divides by zero to produce an exception.
    //-----------------------------------------------------------------
    public static void main (String[] args)
    {
        int numerator = 10;
        int denominator = 0;

        System.out.println ("Before the attempt to divide by zero.");
        try{
            System.out.println (numerator / denominator);
        }catch (ArithmeticException arex) {
            System.out.println ("Attempt to divide by zero. Not good.");
        }
        System.out.println ("This text will NOW be printed.");
    }
}
```

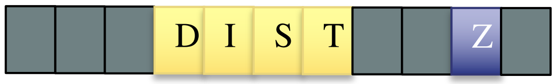
```
Before the attempt to divide by zero.
Attempt to divide by zero. Not good.
This text will NOW be printed.
```

10.3 – The try Statement

- Exceptions are ***thrown*** by a program, and may be ***caught*** and ***handled*** by another part of the program
- To handle an exception, the line that throws the exception is executed within a ***try block***
- A try block is followed by one or more ***catch clauses***
- When an exception occurs, processing continues at the first catch clause that matches the exception type

```
// here is code that
// should generate no exceptions
try {
       // code to monitor
       // several possible things
       // that can go wrong
       // goes here
}
catch (ExceptionTypeA ex) {
       //handler for ExceptionTypeA
}
catch (ExceptionTypeB ex) {
       //handler for ExceptionTypeB
}
// after a catch, continue here
```

Using Exceptions in an “exceptional” way ;-)



// Counts the number of product codes that are entered
// with a zone of R and district greater than 2000.

```
        zone = code.charAt(9);
        district = Integer.parseInt(code.substring(3, 7));
        valid++;
        if (zone == 'R' && district > 2000) banned++;
```

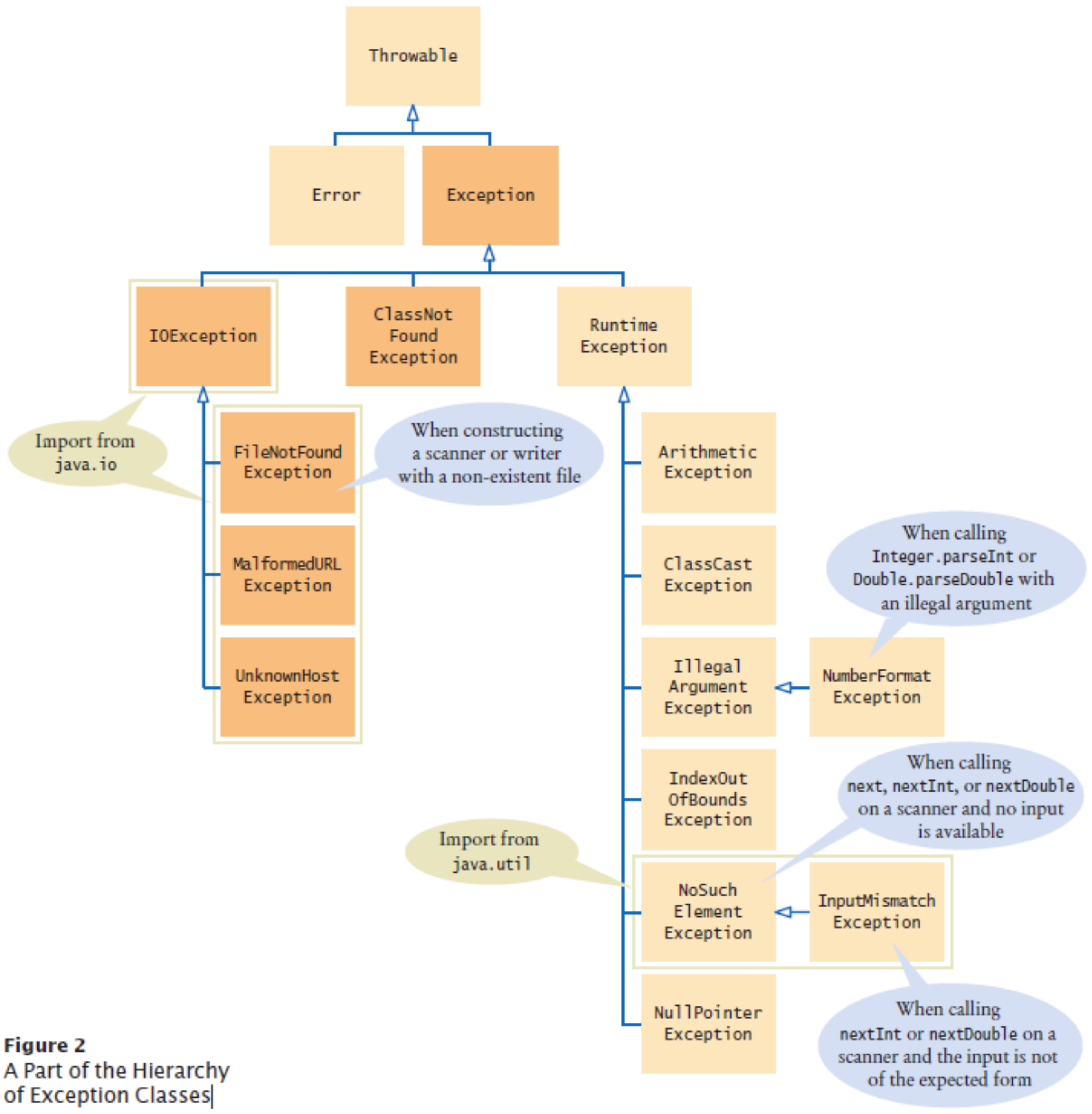
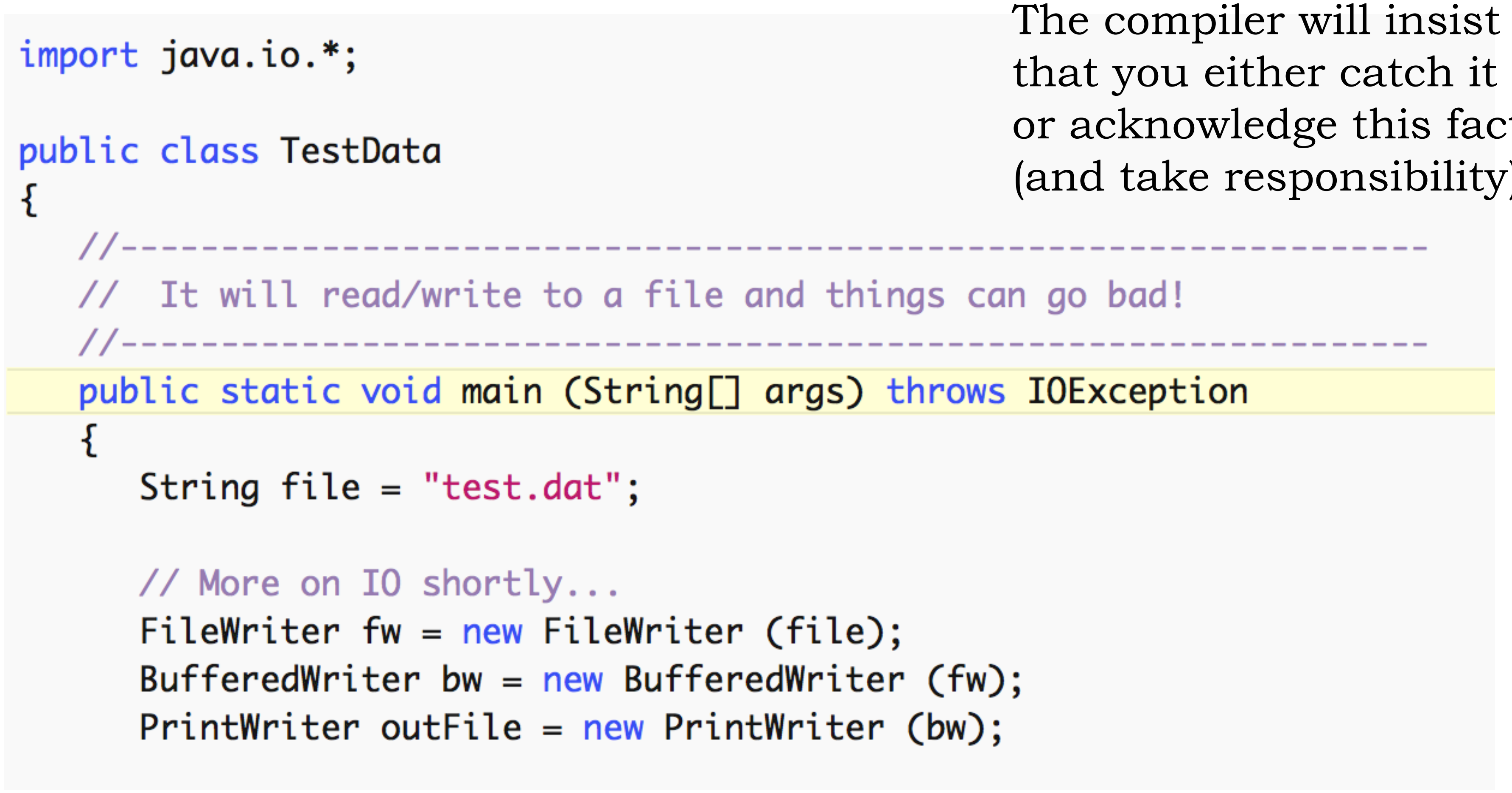

Throwable
Error
Exception
IOException
ClassNot Found Exception
Runtime Exception
Import from java.io
FileNotFound Exception
When constructing a scanner or writer with a non-existent file
MalformedURL Exception
UnknownHost Exception
Arithmetic Exception
ClassCast Exception
When calling Integer.parseInt or Double.parseDouble with an illegal argument
Illegal Argument Exception
NumberFormat Exception
IndexOut OfBounds Exception
When calling next, nextInt, or nextDouble on a scanner and no input is available
Import from java.util
NoSuch Element Exception
InputMismatch Exception
NullPointer Exception
When calling nextInt or nextDouble on a scanner and the input is not of the expected form

Figure 2
A Part of the Hierarchy of Exception Classes

The `throws` clause

Everyone knows that I/O is unpredictable and can throw an exception.

The compiler will insist that you either catch it or acknowledge this fact (and take responsibility).



```java
import java.io.*;

public class TestData
{
   //-----------------------------------------------------------------
   //  It will read/write to a file and things can go bad!
   //-----------------------------------------------------------------
   public static void main (String[] args) throws IOException
   {
      String file = "test.dat";

      // More on IO shortly...
      FileWriter fw = new FileWriter (file);
      BufferedWriter bw = new BufferedWriter (fw);
      PrintWriter outFile = new PrintWriter (bw);
```

10.5 – An exception is either *checked* or *unchecked*

- A ***checked*** *exception* requires explicit handling. It **must**

or

- The compiler will issue error if a checked exception is **not caught** or **asserted** in a throws clause

- An ***unchecked*** *exception* does not require explicit handling (but try to catch)
- The only unchecked Java exceptions are objects of type **RuntimeException** (or any of its descendants)

- **Errors** are similar to RuntimeException and its descendants in the sense that
  - Errors cannot be caught
  - Errors do not require a throws clause

I/O with Scanner and PrintWriter

Great Resource!

Learn and Reuse!

Reading in from the keyboard

```
/* Read in lines of text from the keyboard,
 * and print out each line after it is read in.
 * Stop when the user hits CONTROL-D.
 */
public static void displayKeyboardInput () {
       // will not throw
       Scanner keyboardScan = new Scanner (System.in);
       do {
            String line = keyboardScan.nextLine();
            System.out.println(line);
         } while (keyboardScan.hasNext());
       keyboardScan.close();
}
```

Replace this as you wish

Reading in from a file

```java
/* Read in the contents of a file line by line,
 * and print out each line after it is read in.
 * Stop when the end of the file is reached.
 */
public static void displayFile (String inFileName) {
   try {
       Scanner fileScan = new Scanner (new File(inFileName));
       while (fileScan.hasNext()) {
           String line = fileScan.nextLine();
           System.out.println(line);
         }
       fileScan.close();
   } catch (IOException ex) {
       System.out.println(ex);
   }
}
```

Replace this as you wish

Reading in from a Web page

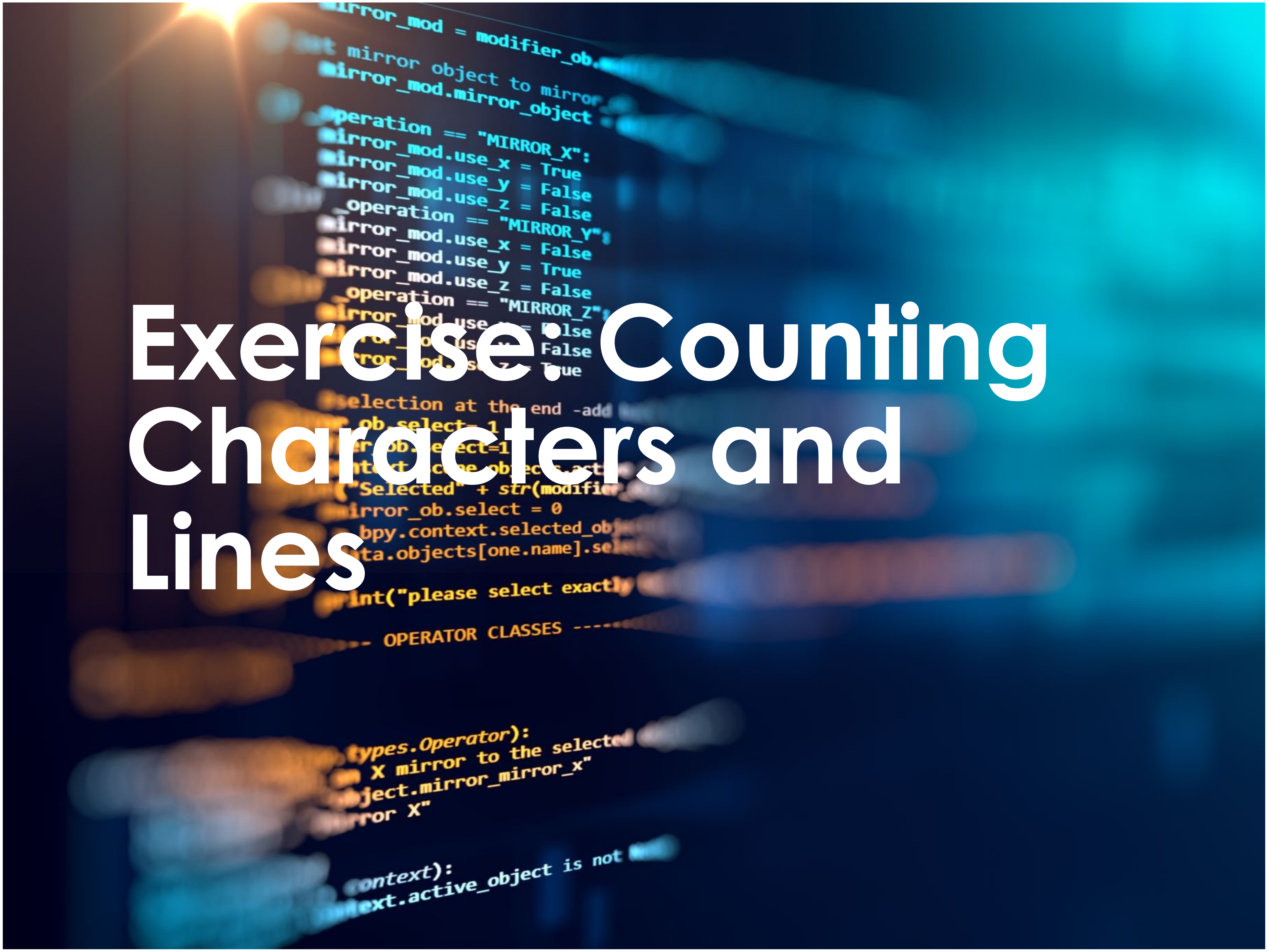
```
/* Read in the contents of a web page line by line,
 * and print out each line after it is read in.
 * Stop when the end of the web page is reached.
 */
public static void displayWebPage (String urlName) {
  try {
      URL u = new URL(urlName);
      Scanner urlScan = new Scanner( u.openStream() );
       while (urlScan.hasNext()) {
           String line = urlScan.nextLine();
           System.out.println(line);
      }
      urlScan.close();
  } catch (IOException ex) {
     System.out.println(ex);
  }
}
```

Replace this as you wish

Writing to a File

```
/* Copies an input file to an output file. Displays an
 * error message if the output file cannot be created.
 */
public static void copyFile(String inFileName,
                             String outFileName){
  try{
    Scanner reader = new Scanner (new File(inFileName));
    PrintWriter writer = new PrintWriter (new File(outFileName));
    while (reader.hasNext()) {
       String input = reader.nextLine());
       writer.println(input);
    }
    writer.close();
    reader.close();
  }catch (IOException ex) {
    System.out.println(ex); // Handle file-not-found
  }
}
```

Replace this as you wish



Exercise: Counting Characters and Lines

Counting Characters and Lines

Write a method that
takes the name of a file as input and
prints out the number of characters in the file and
the number of lines in the file.

```
public static void countCharsAndLines(String filename) {





}
```