AP Computer Science B

Java Object-Oriented Programming [Ver. 3.0]

Unit 4: Object-Oriented Programming

CHAPTER 14: DATA PROCESSING

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

- What is data processing and file processing?
- Data Set: Data Clearing, Data set, Machine Learning, Data Visualization.
- File I/O and data set.
- Object-Oriented Programming for Data Science
- Exception Handling.

Overview

LECTURE 1

Data Science

- Data science combines math and statistics, specialized programming, advanced analytics, artificial intelligence (AI), and machine learning with specific subject matter expertise to uncover actionable insights hidden in an organization’s data.
- These insights can be used to guide **decision making** and **strategic planning**.

Topics

- Qualitative versus quantitative data
- Logical views and physical views
- Abstract Data Types
- Data Structures

QUALITATIVE VS. QUANTITATIVE

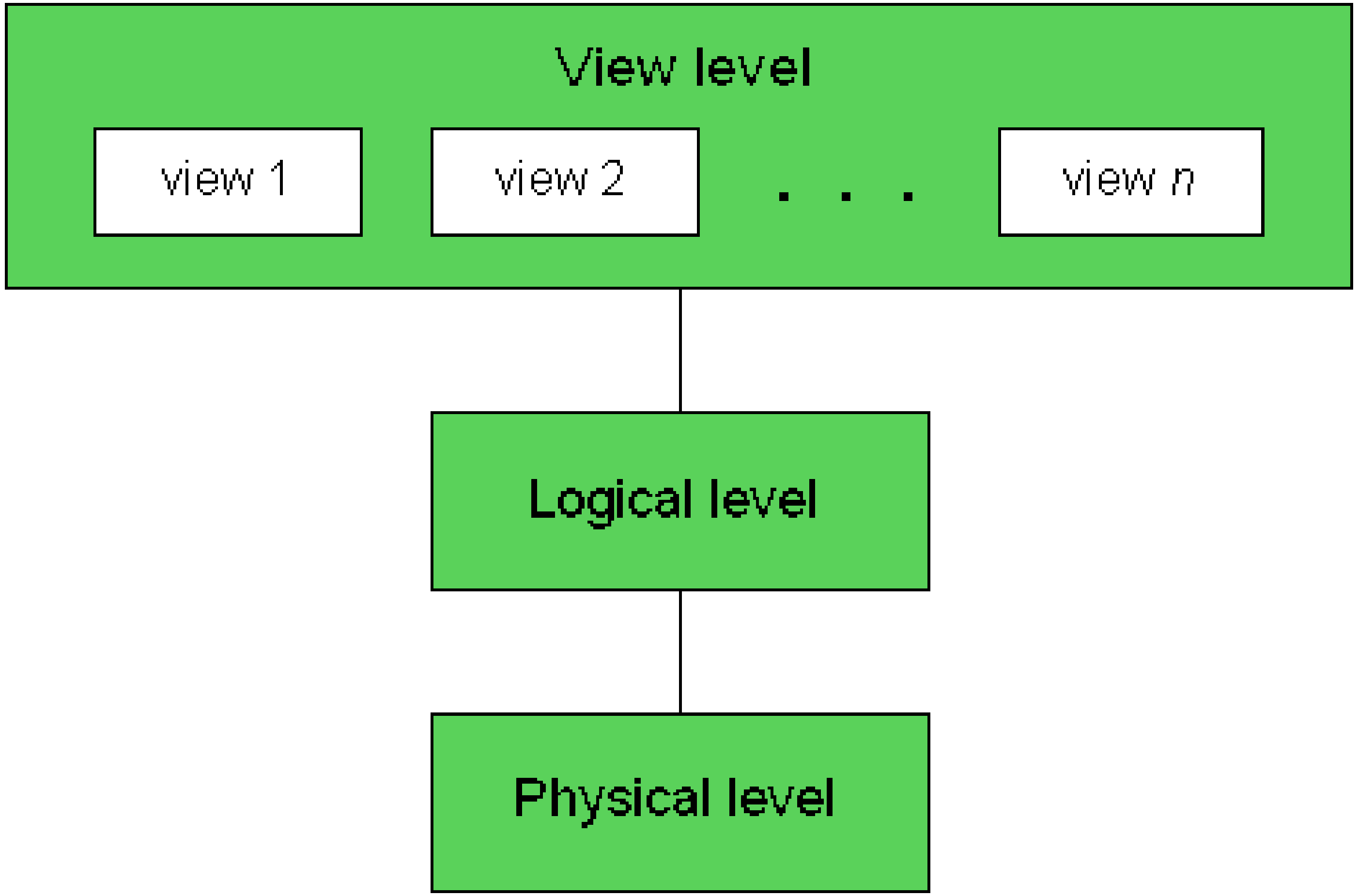
Data Collection

from interview sessions, discussion with focus groups, case studies, and ethnography

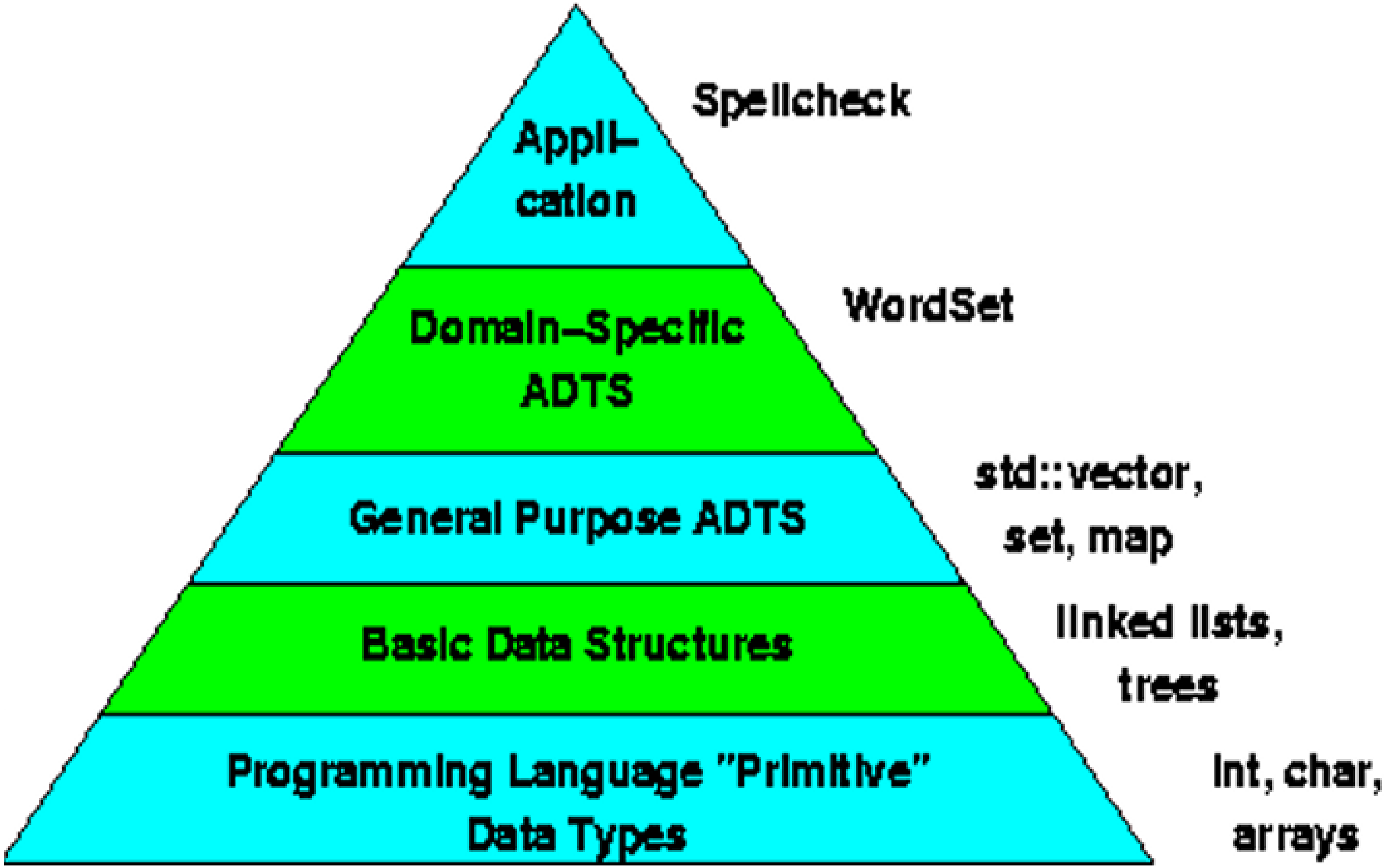
Data Collection

from surveys, experiments, observations, and content analysis.

Data Abstraction Levels



Abstraction Data Types



Application — Spellcheck

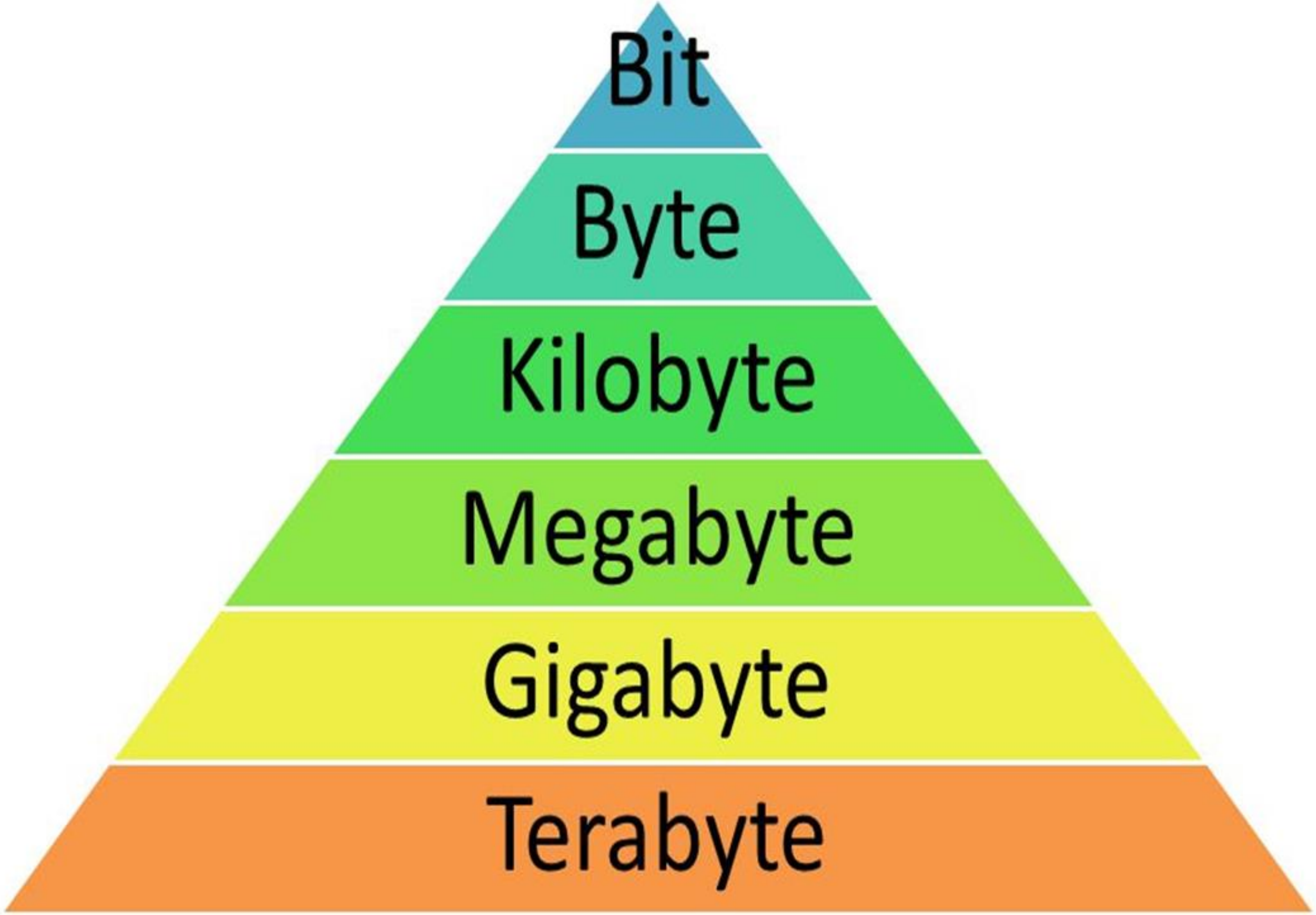
Domain-Specific ADTS — WordSet

General Purpose ADTS — std::vector, set, map

Basic Data Structures — linked lists, trees

Programming Language "Primitive" Data Types — int, char, arrays

Types of Units of Storage



Bit

Byte

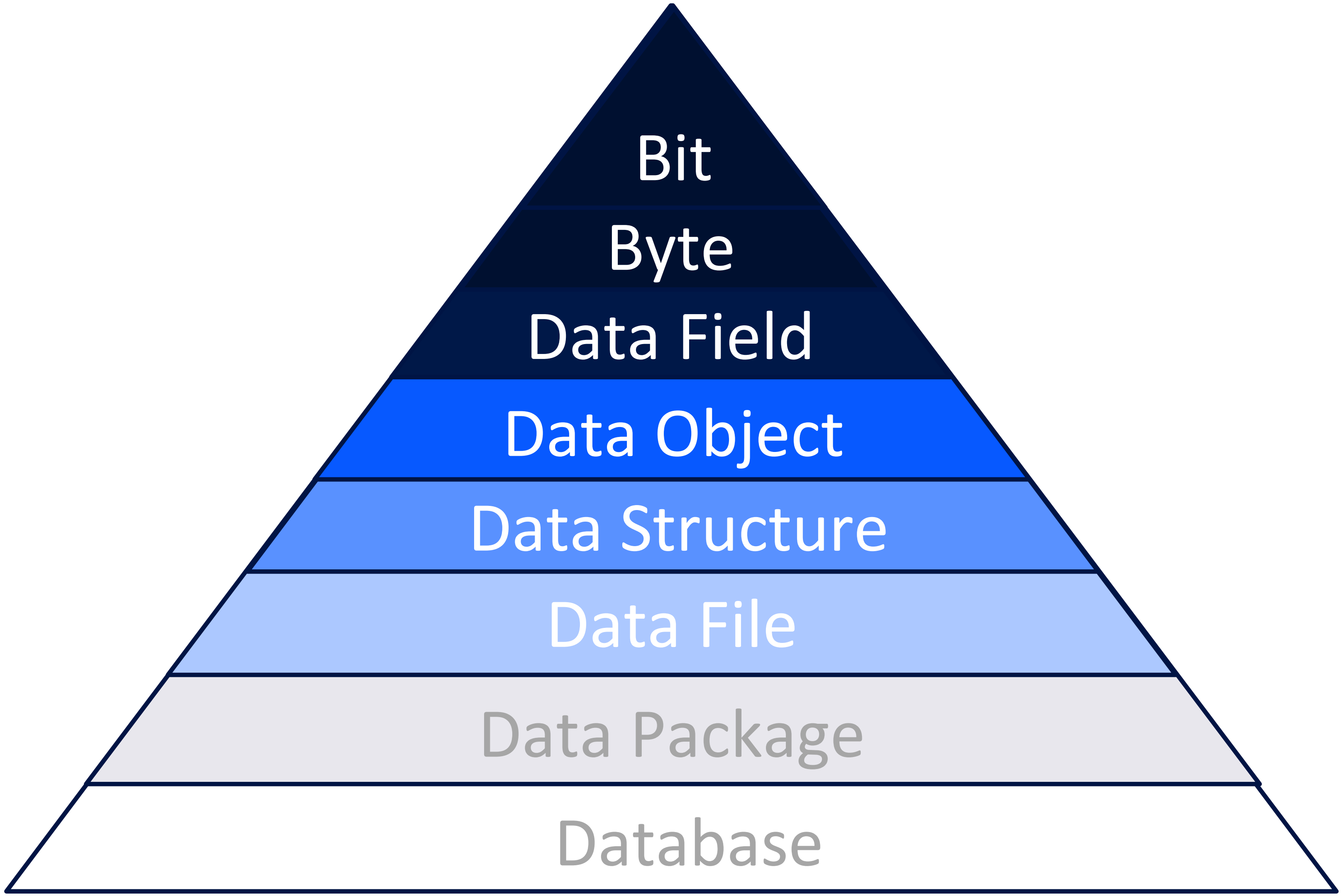
Kilobyte

Megabyte

Gigabyte

Terabyte

Data Structures



Bit

Byte

Data Field

Data Object

Data Structure

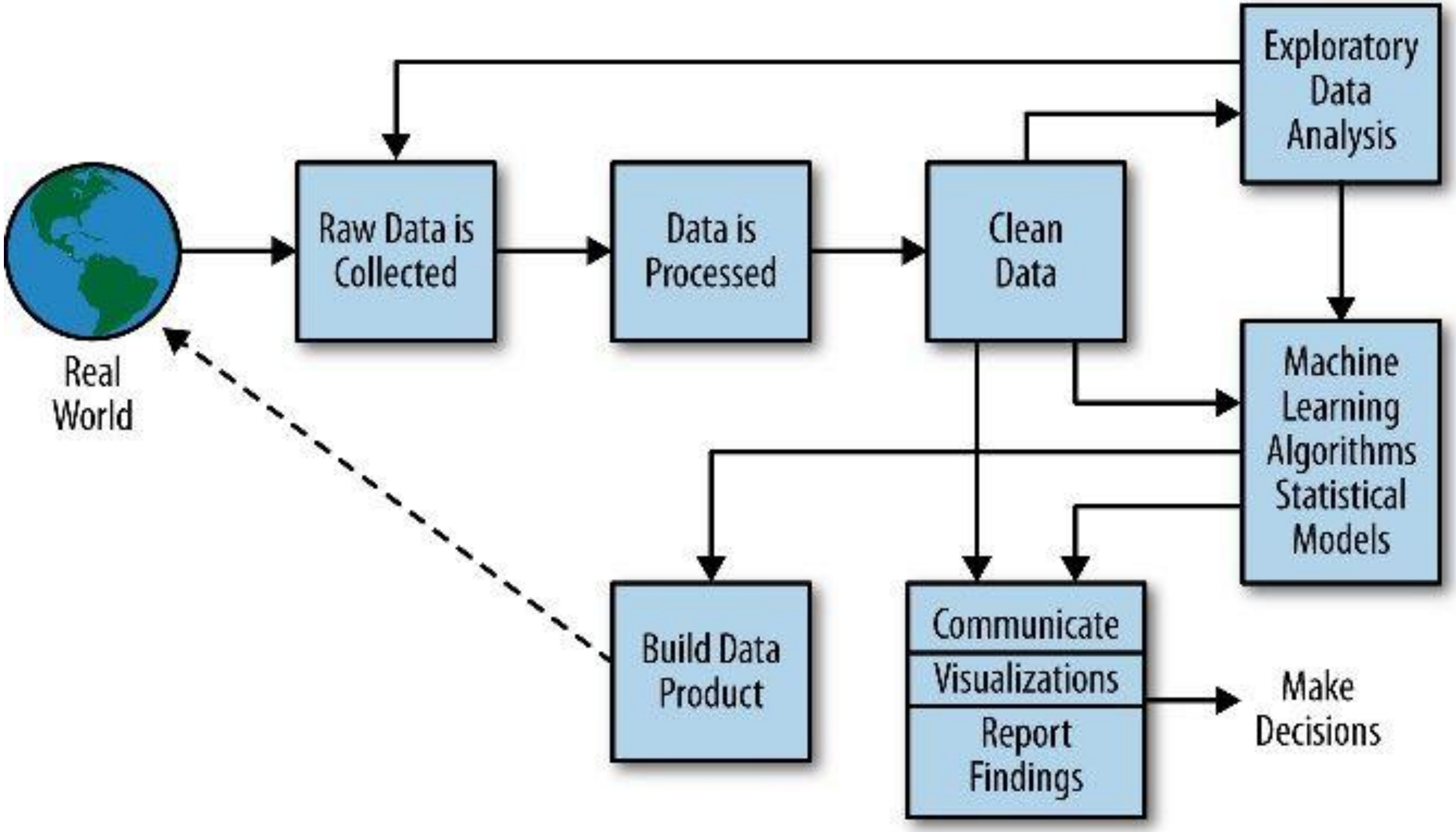
Data File

Data Package

Database

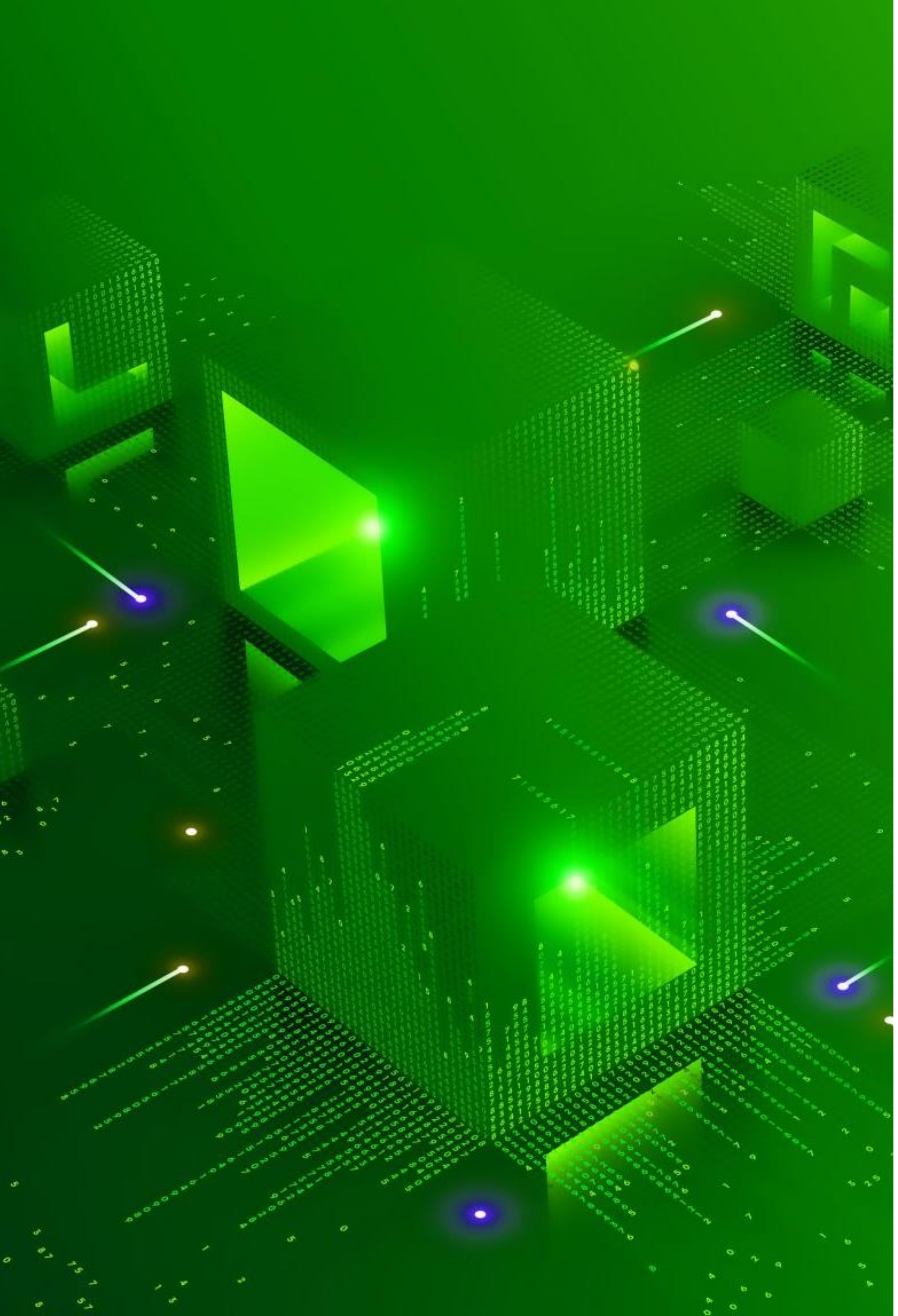
Data Processing

LECTURE 2


Real World
Raw Data is Collected
Data is Processed
Clean Data
Exploratory Data Analysis
Machine Learning Algorithms Statistical Models
Build Data Product
Communicate
Visualizations
Report Findings
Make Decisions

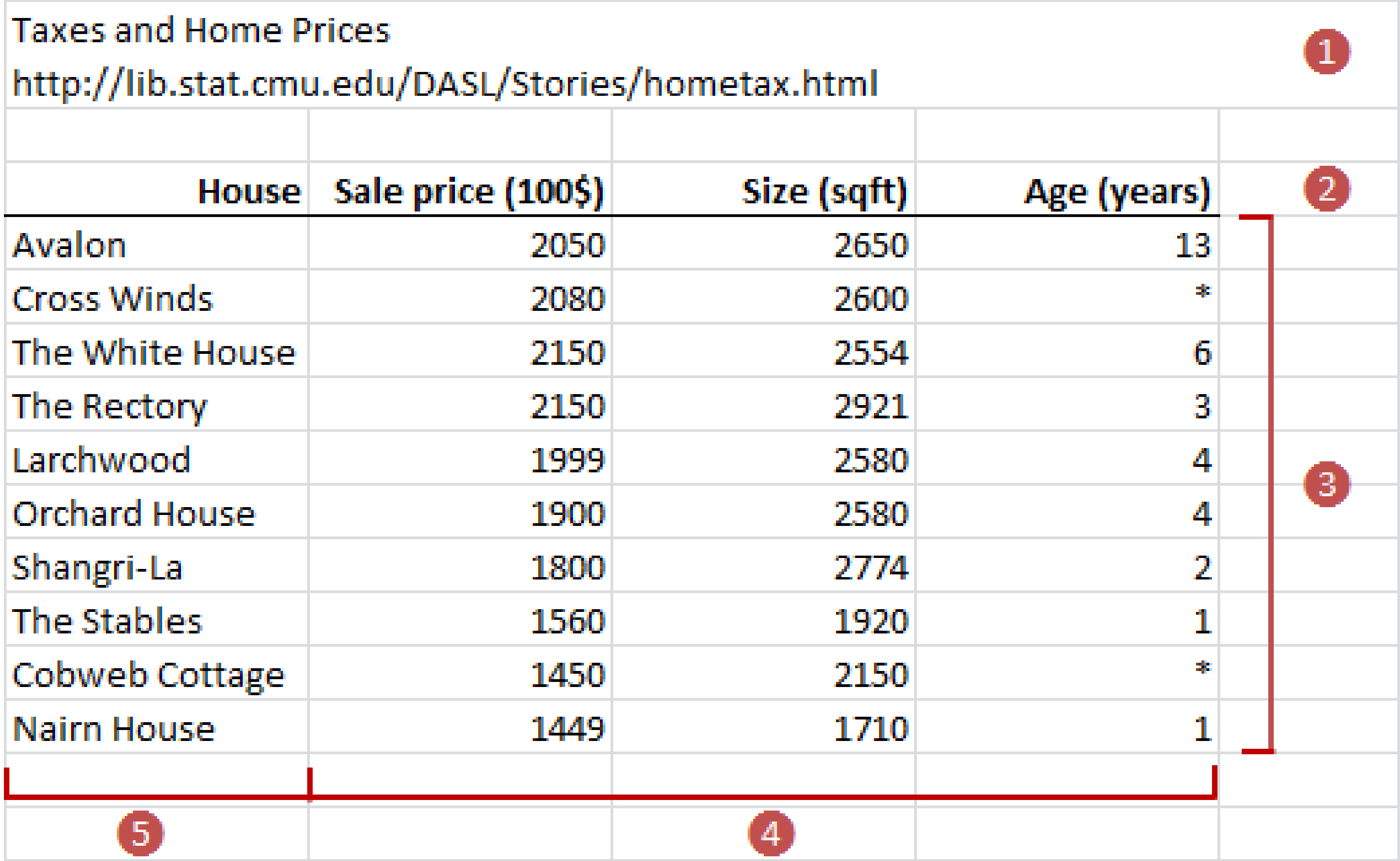
Data Set

LECTURE 3



Data Set

- A data set is a collection of related, discrete items of related data that may be accessed individually or in combination or managed as a whole entity.
- A data set is organized into some type of data structure.



Taxes and Home Prices

http://lib.stat.cmu.edu/DASL/Stories/hometax.html

House	Sale price (100$)	Size (sqft)	Age (years)
Avalon	2050	2650	13
Cross Winds	2080	2600	*
The White House	2150	2554	6
The Rectory	2150	2921	3
Larchwood	1999	2580	4
Orchard House	1900	2580	4
Shangri-La	1800	2774	2
The Stables	1560	1920	1
Cobweb Cottage	1450	2150	*
Nairn House	1449	1710	1

Demonstration Program

EXCEL DATA SET

File Class

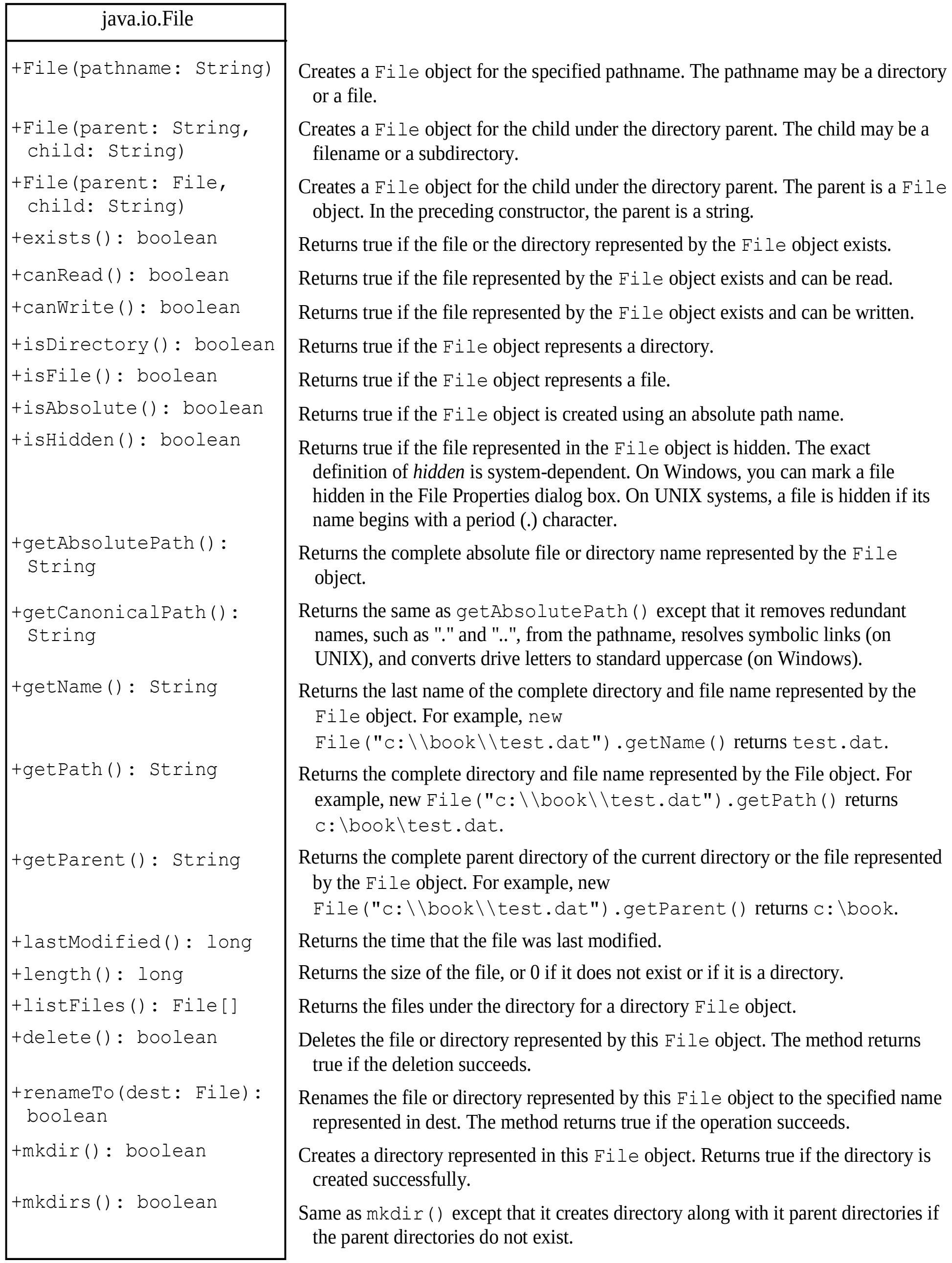
LECTURE 4

File Class

- The File class is intended to provide an abstraction that deals with most of the **machine-dependent** complexities of files and path names in a machine-independent fashion. The **filename** is a **string**. The File class is a wrapper class for the file name and its directory path.

File Class

- Obtaining file properties and manipulating file



java.io.File	
`+File(pathname: String)`	Creates a `File` object for the specified pathname. The pathname may be a directory or a file.
`+File(parent: String, child: String)`	Creates a `File` object for the child under the directory parent. The child may be a filename or a subdirectory.
`+File(parent: File, child: String)`	Creates a `File` object for the child under the directory parent. The parent is a `File` object. In the preceding constructor, the parent is a string.
`+exists(): boolean`	Returns true if the file or the directory represented by the `File` object exists.
`+canRead(): boolean`	Returns true if the file represented by the `File` object exists and can be read.
`+canWrite(): boolean`	Returns true if the file represented by the `File` object exists and can be written.
`+isDirectory(): boolean`	Returns true if the `File` object represents a directory.
`+isFile(): boolean`	Returns true if the `File` object represents a file.
`+isAbsolute(): boolean`	Returns true if the `File` object is created using an absolute path name.
`+isHidden(): boolean`	Returns true if the file represented in the `File` object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On UNIX systems, a file is hidden if its name begins with a period (.) character.
`+getAbsolutePath(): String`	Returns the complete absolute file or directory name represented by the `File` object.
`+getCanonicalPath(): String`	Returns the same as `getAbsolutePath()` except that it removes redundant names, such as "." and "..", from the pathname, resolves symbolic links (on UNIX), and converts drive letters to standard uppercase (on Windows).
`+getName(): String`	Returns the last name of the complete directory and file name represented by the `File` object. For example, `new File("c:\\book\\test.dat").getName()` returns `test.dat`.
`+getPath(): String`	Returns the complete directory and file name represented by the File object. For example, new `File("c:\\book\\test.dat").getPath()` returns `c:\book\test.dat`.
`+getParent(): String`	Returns the complete parent directory of the current directory or the file represented by the `File` object. For example, `new File("c:\\book\\test.dat").getParent()` returns `c:\book`.
`+lastModified(): long`	Returns the time that the file was last modified.
`+length(): long`	Returns the size of the file, or 0 if it does not exist or if it is a directory.
`+listFiles(): File[]`	Returns the files under the directory for a directory `File` object.
`+delete(): boolean`	Deletes the file or directory represented by this `File` object. The method returns true if the deletion succeeds.
`+renameTo(dest: File): boolean`	Renames the file or directory represented by this `File` object to the specified name represented in dest. The method returns true if the operation succeeds.
`+mkdir(): boolean`	Creates a directory represented in this `File` object. Returns true if the directory is created successfully.
`+mkdirs(): boolean`	Same as `mkdir()` except that it creates directory along with it parent directories if the parent directories do not exist.

Class java.io.File

The class java.io.File can represent either a file or a directory.

A **path string** is used to locate a file or a directory. Unfortunately, path strings are system dependent, e.g., **"c:\myproject\java\Hello.java"** in Windows or **"/myproject/java/Hello.java"** in Unix/Mac.

- Windows use back-slash '\' as the directory separator; while Unixes/Mac use forward-slash '/'.
- Windows use semi-colon ';' as path separator to separate a list of paths; while Unixes/Mac use colon ':'.
- Windows use "\r\n" as line delimiter for text file; while Unixes use "\n" and Mac uses "\r". **[ASCII 10(\n)/13(\r)]**
- The "c:\" or "\" is called the root. Windows supports multiple roots, each maps to a drive (e.g., "c:\", "d:\"). Unixes/Mac has a single root ("\").

File Path

A path could be **absolute** (beginning from the root) or **relative** (which is relative to a reference directory). Special notations "." and ".." denote the current directory and the parent directory, respectively.

The java.io.File class maintains these system-dependent properties, for you to write programs that are portable:

- **Directory Separator:** in static fields File.separator (as String) and File.separatorChar. [They failed to follow the Java naming convention for constants adopted since JDK 1.2.] As mentioned, Windows use backslash '\'; while Unixes/Mac use forward slash '**/**'.
- **Path Separator:** in static fields File.pathSeparator (as String) and File.pathSeparatorChar. As mentioned, Windows use semi-colon ';' to separate a list of paths; while Unixes/Mac use colon ':'.

File Access Using URL Locator

You can construct a `File` instance with a path string or URI, as follows. Take note that the physical file/directory may or may not exist. A file URL takes the form of `file://...`, e.g., `file:///d:/docs/programming/java/test.html`.

```
public File(String pathString)
public File(String parent, String child)
public File(File parent, String child) // Constructs a File instance based on the given path string.
public File(URI uri) // Constructs a File instance by converting from the given file-URI "file://...."
```

For examples,

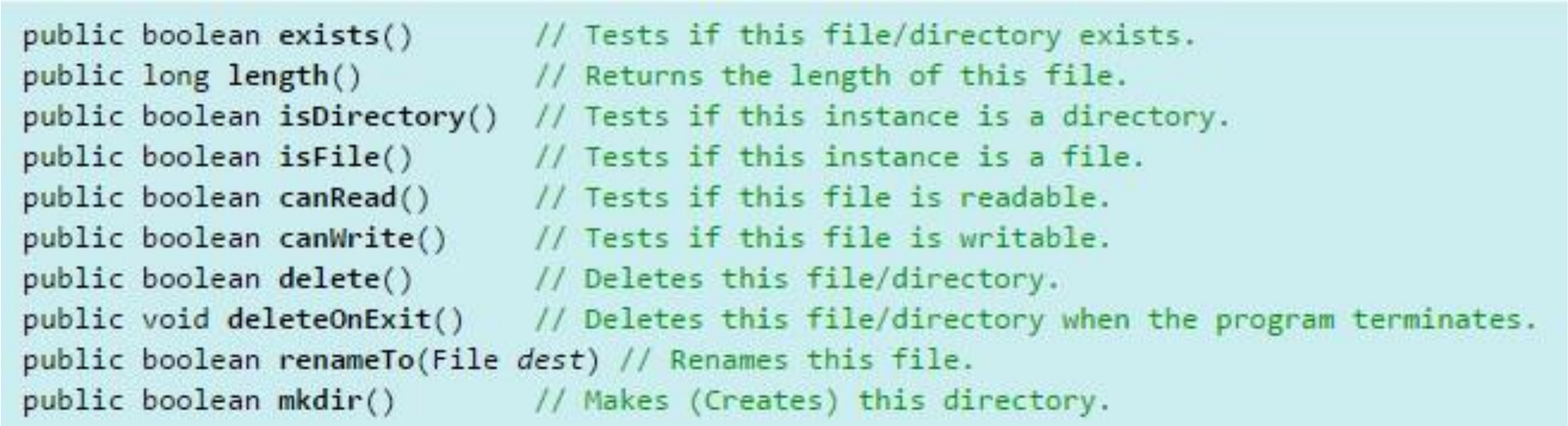
```
File file = new File("in.txt"); // A file relative to the current working directory
File file = new File("d:\\myproject\\java\\Hello.java");
// A file with absolute path
File dir = new File("c:\\temp"); // A directory
```

For applications that you intend to distribute as JAR files, you should use the `URL` class to reference the resources, as it can reference disk files as well as JAR'ed files , for example,

```
java.net.URL url = this.getClass().getResource("icon.png");
```

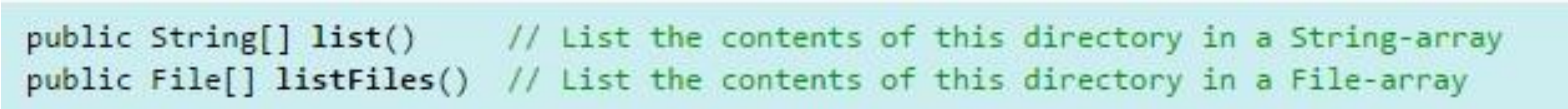
Methods to Verify a File/Directory and Lists Directory Files

Verify a File/Directory:



```
public boolean exists()        // Tests if this file/directory exists.
public long length()           // Returns the length of this file.
public boolean isDirectory()   // Tests if this instance is a directory.
public boolean isFile()        // Tests if this instance is a file.
public boolean canRead()       // Tests if this file is readable.
public boolean canWrite()      // Tests if this file is writable.
public boolean delete()        // Deletes this file/directory.
public void deleteOnExit()     // Deletes this file/directory when the program terminates.
public boolean renameTo(File dest) // Renames this file.
public boolean mkdir()         // Makes (Creates) this directory.
```

List Directory:



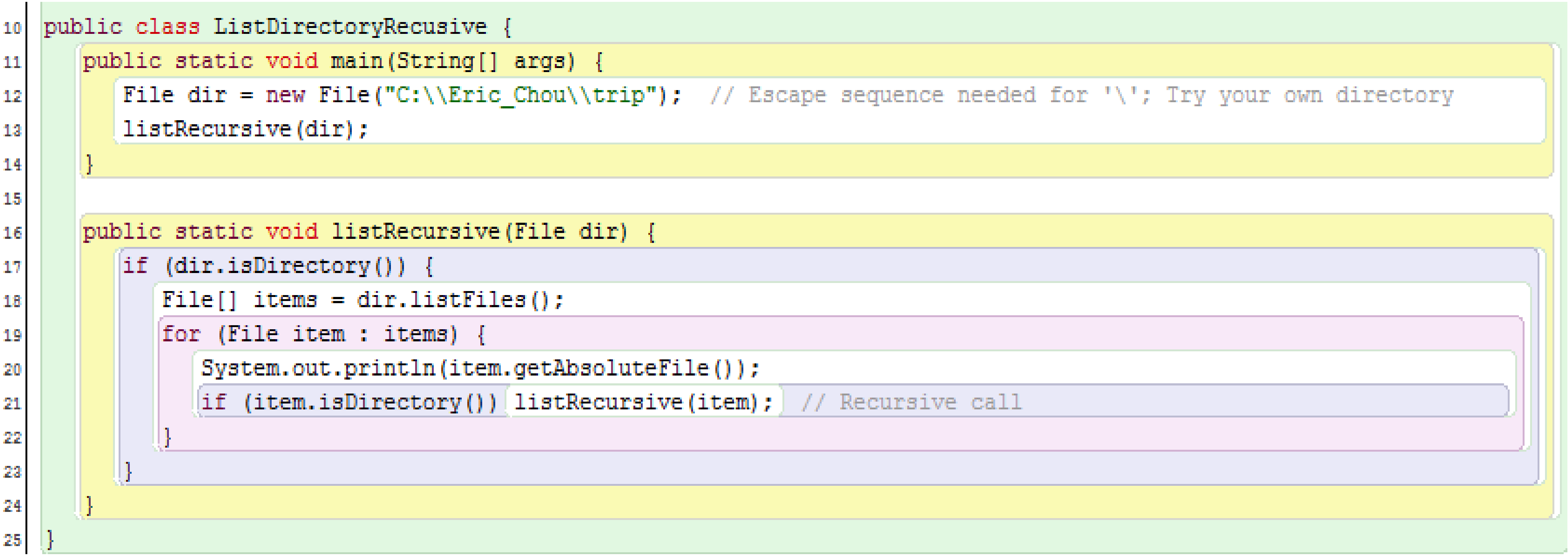
```
public String[] list()      // List the contents of this directory in a String-array
public File[] listFiles()   // List the contents of this directory in a File-array
```

Demonstration Program

LISTDIRECTORYRECUSIVE.JAVA

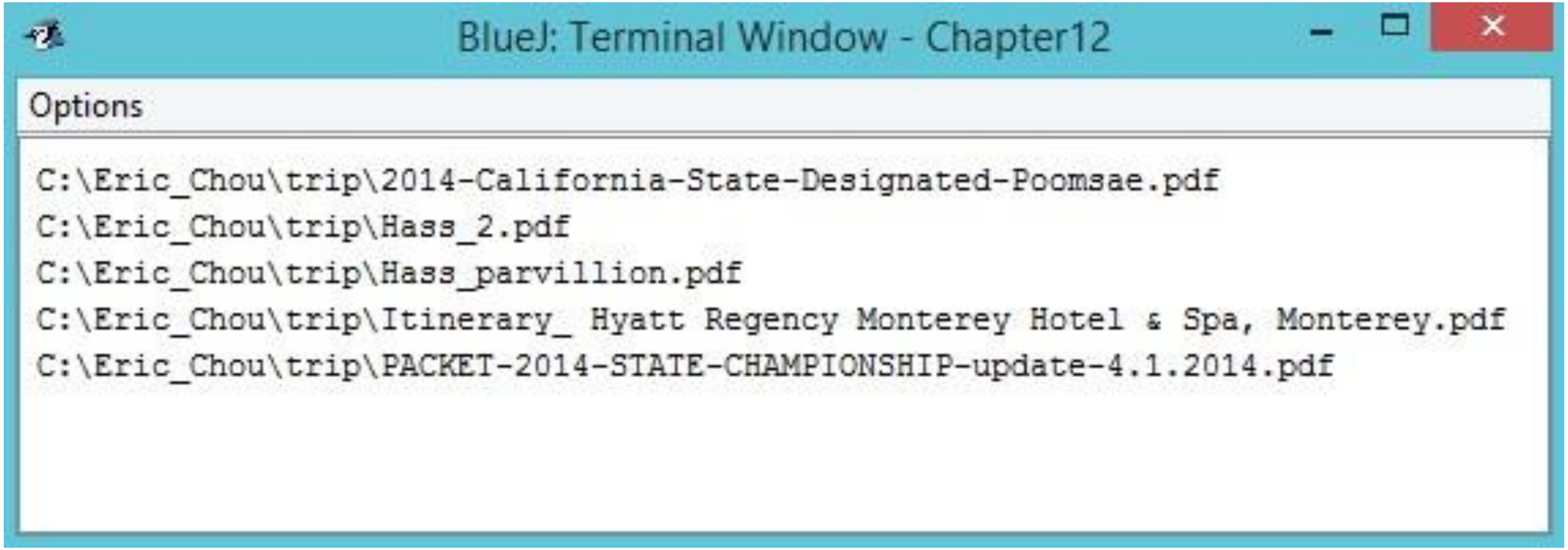
Demo Program:

The following program recursively lists the contents of a given directory (similar to Unix's "ls -r" command).



```java
public class ListDirectoryRecusive {
    public static void main(String[] args) {
        File dir = new File("C:\\Eric_Chou\\trip");  // Escape sequence needed for '\'; Try your own directory
        listRecursive(dir);
    }

    public static void listRecursive(File dir) {
        if (dir.isDirectory()) {
            File[] items = dir.listFiles();
            for (File item : items) {
                System.out.println(item.getAbsoluteFile());
                if (item.isDirectory()) listRecursive(item);  // Recursive call
            }
        }
    }
}
```

Results:



```
C:\Eric_Chou\trip\2014-California-State-Designated-Poomsae.pdf
C:\Eric_Chou\trip\Hass_2.pdf
C:\Eric_Chou\trip\Hass_parvillion.pdf
C:\Eric_Chou\trip\Itinerary_ Hyatt Regency Monterey Hotel & Spa, Monterey.pdf
C:\Eric_Chou\trip\PACKET-2014-STATE-CHAMPIONSHIP-update-4.1.2014.pdf
```

Demo Program: Explore File Properties

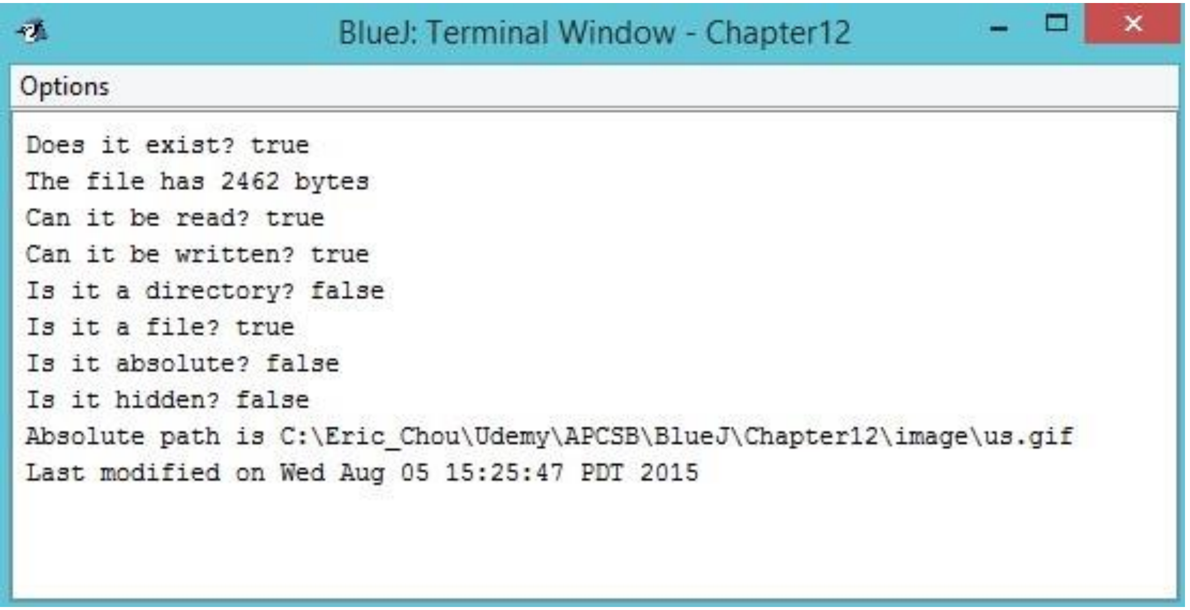
TestFileClass.java

Use a program that demonstrates how to create files in a platform-independent way and use the methods in the File class to obtain their properties.

Demonstration Program

TESTFILECLASS.JAVA

TestFileClass



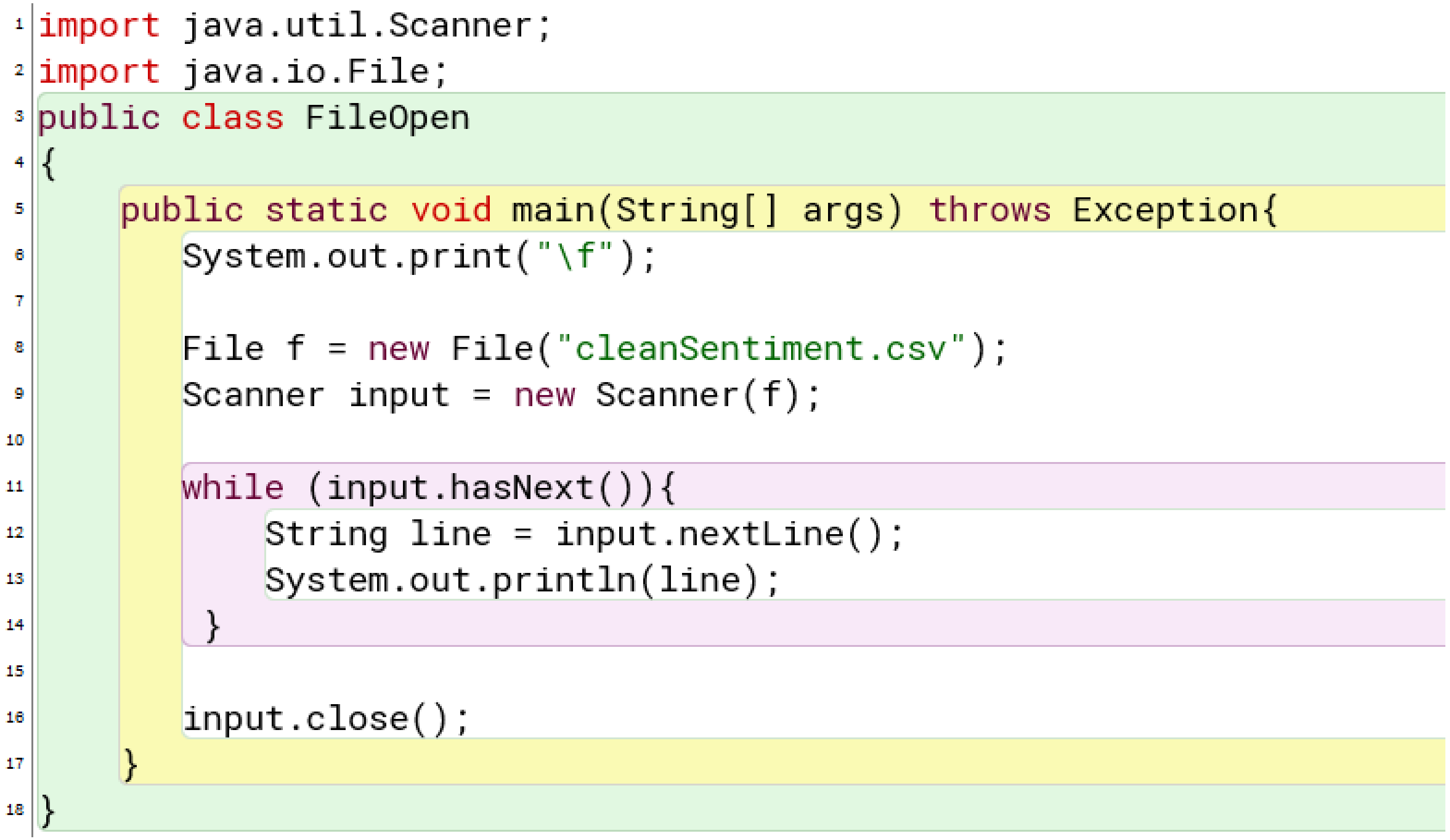
```
Does it exist? true
The file has 2462 bytes
Can it be read? true
Can it be written? true
Is it a directory? false
Is it a file? true
Is it absolute? false
Is it hidden? false
Absolute path is C:\Eric_Chou\Udemy\APCSB\BlueJ\Chapter12\image\us.gif
Last modified on Wed Aug 05 15:25:47 PDT 2015
```

```
public class TestFileClass {
  public static void main(String[] args) {
    java.io.File file = new java.io.File("image/us.gif");
    System.out.println("Does it exist? " + file.exists());
    System.out.println("The file has " + file.length() + " bytes");
    System.out.println("Can it be read? " + file.canRead());
    System.out.println("Can it be written? " + file.canWrite());
    System.out.println("Is it a directory? " + file.isDirectory());
    System.out.println("Is it a file? " + file.isFile());
    System.out.println("Is it absolute? " + file.isAbsolute());
    System.out.println("Is it hidden? " + file.isHidden());
    System.out.println("Absolute path is " +
      file.getAbsolutePath());
    System.out.println("Last modified on " +
      new java.util.Date(file.lastModified()));
  }
}
```

File Input

LECTURE 5

FileOpen.java

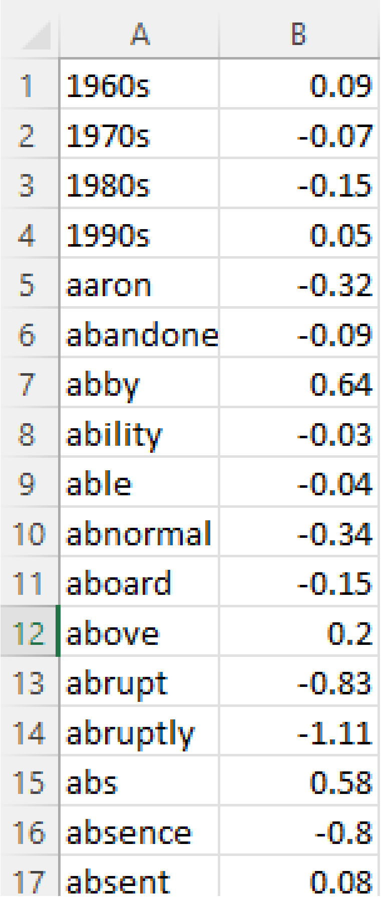


```java
import java.util.Scanner;
import java.io.File;
public class FileOpen
{
    public static void main(String[] args) throws Exception{
        System.out.print("\f");

        File f = new File("cleanSentiment.csv");
        Scanner input = new Scanner(f);

        while (input.hasNext()){
            String line = input.nextLine();
            System.out.println(line);
         }

        input.close();
    }
}
```

AP Consumer Review Lab

- The Consumer Review Lab provides students with the opportunity to work on a larger assignment involving multiple classes early in the course, before they necessarily know how to write their own classes. By focusing on calling methods and using the control structures of selection and iteration, students will have practice that directly relates to the "Methods and Control Structures" free-response question type that they will see on the end-of-year exam.
- Because students often struggle with methods in the **String class**, this lab specifically focuses on calling these methods and building strings in an engaging and interesting way.

Data Set – cleanSentiment.csv



	A	B
1	1960s	0.09
2	1970s	-0.07
3	1980s	-0.15
4	1990s	0.05
5	aaron	-0.32
6	abandone	-0.09
7	abby	0.64
8	ability	-0.03
9	able	-0.04
10	abnormal	-0.34
11	aboard	-0.15
12	above	0.2
13	abrupt	-0.83
14	abruptly	-1.11
15	abs	0.58
16	absence	-0.8
17	absent	0.08

Sentiment Value

- As a class, look at this cleanSentiment.csv list of words from the lab. Can you find your positive and negative words on the list? Notice that each word has a positive or negative integer value assigned to it. This value is called the **sentiment value** of the word. A large positive sentiment value means that word has appeared in a lot of positive contexts.

Sentiment Value

- The higher the number, the more positive the sentiment. And a large negative sentiment value means that word has appeared in a lot of negative contexts. This list was generated by a computer program that counted the frequency of each word in lots of online reviews that were rated by humans as positive or negative. Do you agree with the sentiment values on the list?
- The quality of the list really depends on the quality and quantity of the data used to generate it.

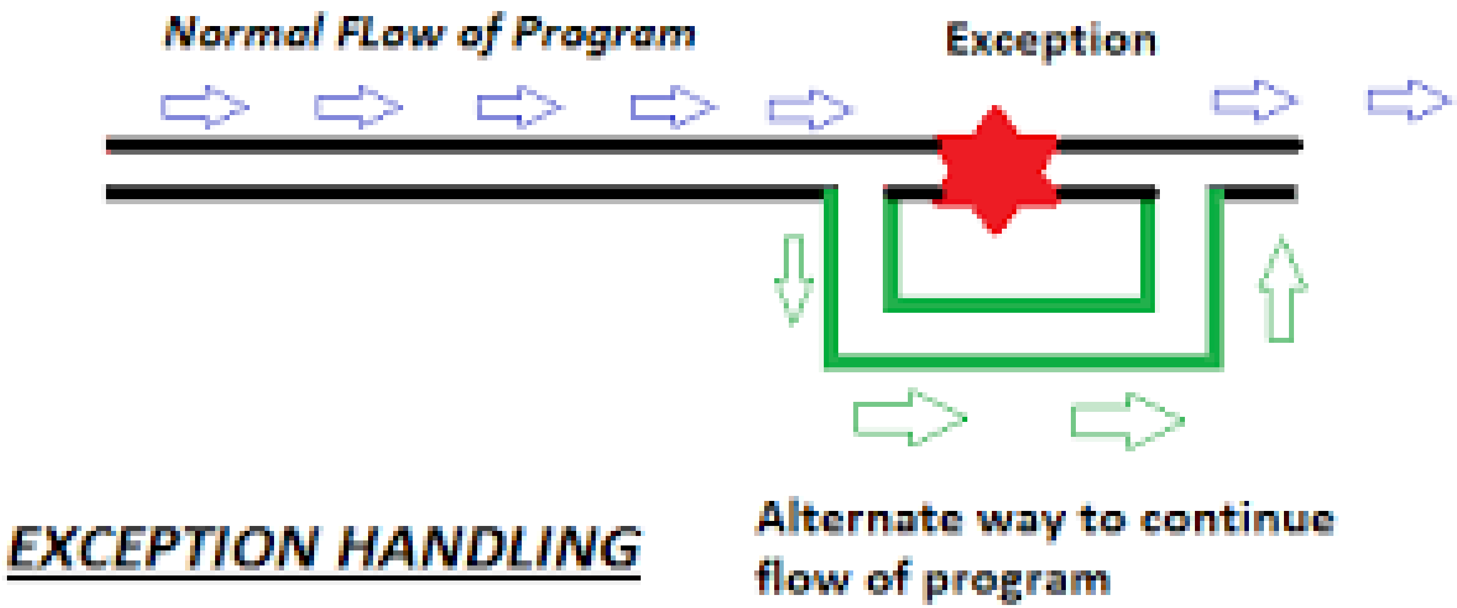
Sentiment Analysis

- The shopping site you used may actually use sentiment analysis to group the reviews into positive and negative reviews for you. Many sites also try to catch fake reviews with **sentiment analysis**. Companies may use sentiment analysis to see if their reviews are more positive or negative and to make improvements to their products or marketing.

Exception

LECTURE 6

Exceptions


Normal FLow of Program
Exception
EXCEPTION HANDLING
Alternate way to continue flow of program

Introduction

Exception Handling enables a program to deal with exceptional situations and continue its normal execution.

- Runtime errors occur while a program is running if the JVM detects an operation that is impossible to carry out.
- For example, if you access an array using an index that is out of bounds, you will get a runtime error with an **ArrayIndexOutOfBoundsException**.
- If you enter a double value when your program expects an integer, you will get a runtime error with an **InputMismatchException**.

Introduction

Exception Handling enables a program to deal with exceptional situations and continue its normal execution.

- In Java, runtime errors are thrown as exceptions. **An exception is an object that represents an error or a condition that prevents execution from proceeding normally.**
- Exception Classes are information classes like Class class returned by getClass().
- If the exception is not handled, the program will terminate abnormally. How can you handle the exception so that the program can continue to run or else terminate gracefully? This chapter introduces this subject and text input and output.

Issues Involves Exceptions

The **exception handling** in java is one of the powerful mechanism to handle the **runtime** errors so that normal flow of the application can be maintained.

- Why it happens and where does it happens?
- How to raise an exception and how to catch it?
- How to handle it?

Exception-Handling Overview Exemplary Run-Time Errors

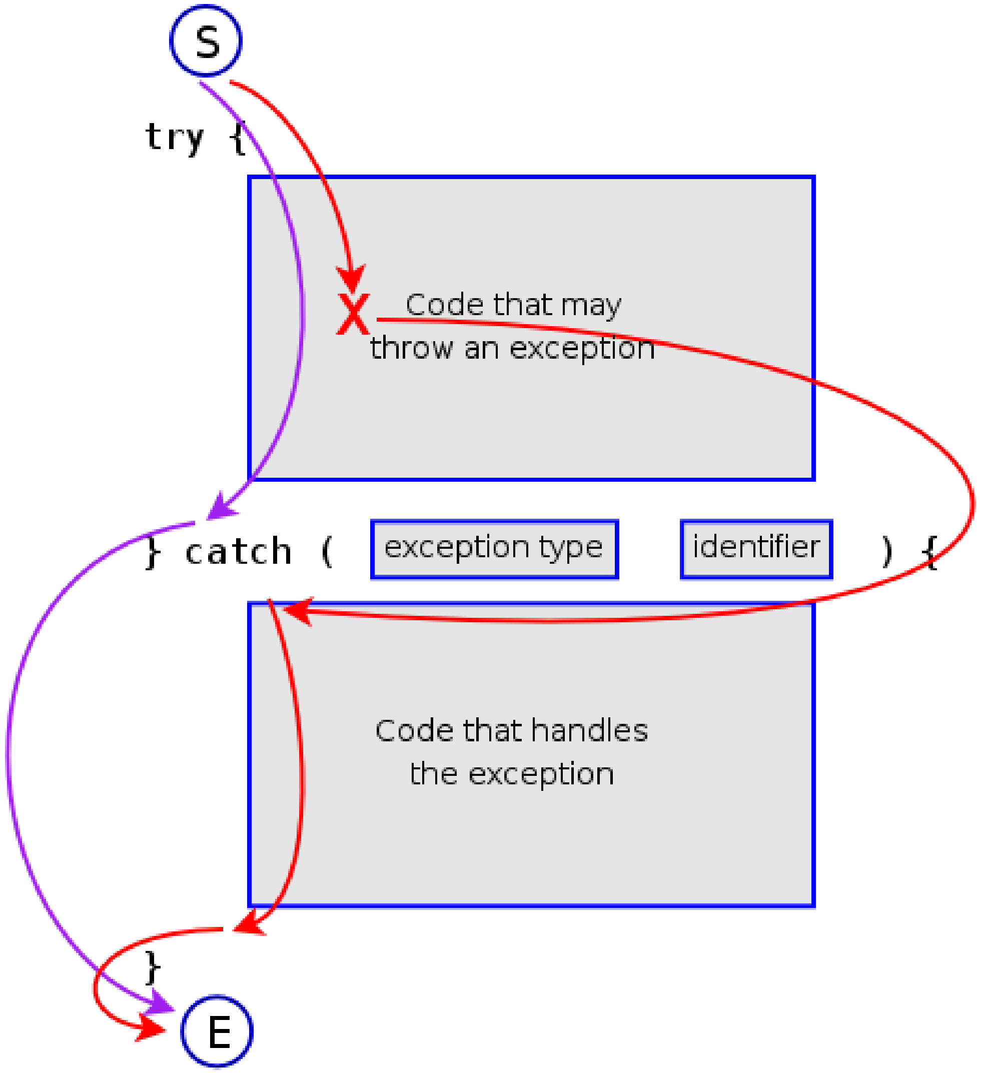
- Show runtime error (**Quotient.java**) occurs when division by 0.
- Fix it using an if statement (**QuotientWithIf.java**) (check and prevent the division by 0 to happen)
- Using method to quarantine the division error (**QuotientWithMethod.java**)
- Introduce try-catch (**QuotientWithException.java**)

Exception Advantages

Handle run-time errors with Exceptions. (**QuotientWithException.java**)

- Now you see the *advantages* of using exception handling. It enables a method to throw an exception to its caller.
- Without this capability, a method must handle the exception or terminate the program.

Exception throws and catch


S
try {
X
Code that may throw an exception
} catch (exception type identifier) {
Code that handles the exception
}
E

- Exception classes are **information** class which contains types of exception and the other information.

From the Method to Exception Handling

- The method **quotient** returns the quotient of two integers. If **number2** is **0**, it cannot return a value, so the program is terminated. This is clearly a problem. You should not let the method terminate the program—the *caller* should decide whether to terminate the program.
- How can a method notify its caller an exception has occurred? Java enables a method to throw an exception that can be caught and handled by the caller. Listing 12.3 can be rewritten, as shown in QuotientWithException.java.

From the Method to Exception Handling

- If **number2** is **0**, the method throws an exception by executing
- **throw new** ArithmeticException(**"Divisor cannot be zero"**);
- The value thrown, in this case **new ArithmeticException("Divisor cannot be zero")**, is called an *exception*. The execution of a **throw** statement is called *throwing an exception*. The exception is an object created from an exception class. In this case, the exception class is **java.lang.ArithmeticException**. The constructor **ArithmeticException(str)** is invoked to construct an exception object, where **str** is a message that describes the exception.

Exception Handling

- When an exception is thrown, the normal execution flow is interrupted. As the name suggests, to “throw an exception” is to pass the exception from one place to another. The statement for invoking the method is contained in a **try** block and a **catch** block. The **try** block contains the code that is executed in normal circumstances. The exception is caught by the **catch** block. The code in the **catch** block is executed to *handle the exception*.

Exception Handling

- Afterward, the statement after the **catch** block is executed. The **throw** statement is analogous to a method call, but instead of calling a method, it calls a **catch** block. In this sense, a **catch** block is like a method definition with a parameter that matches the type of the value being thrown. Unlike a method, however, after the **catch** block is executed, the program control does not return to the **throw** statement; instead, it executes the next statement after the **catch** block.

Exception Handling

like calling a method

The identifier **ex** in the **catch**–block header

```
catch (ArithmeticException ex)
```

acts very much like a parameter in a method. Thus, this parameter is referred to as a **catch**–block parameter. The type (e.g., **ArithmeticException**) preceding **ex** specifies what kind of exception the **catch** block can catch. Once the exception is caught, you can access the thrown value from this parameter in the body of a **catch** block.

Exception Handling

like calling a method

- In summary, a template for a **try**-**throw**-**catch** block may look like this:

```
try {
    Code to run;
    A statement or a method that may throw an exception;
    More code to run;
}
catch (type ex) {
    Code to process the exception;
}
```

- An exception may be thrown directly by using a **throw** statement in a **try** block, or by invoking
- a method that may throw an exception.

Handling InputMismatchException

- Errors when input data type mismatched (**InputMismatchException.java**)
- By handling InputMismatchException, your program will continuously read an input until it is correct.

Demonstration Program

INPUTMISMATCHEXCEPTION.JAVA

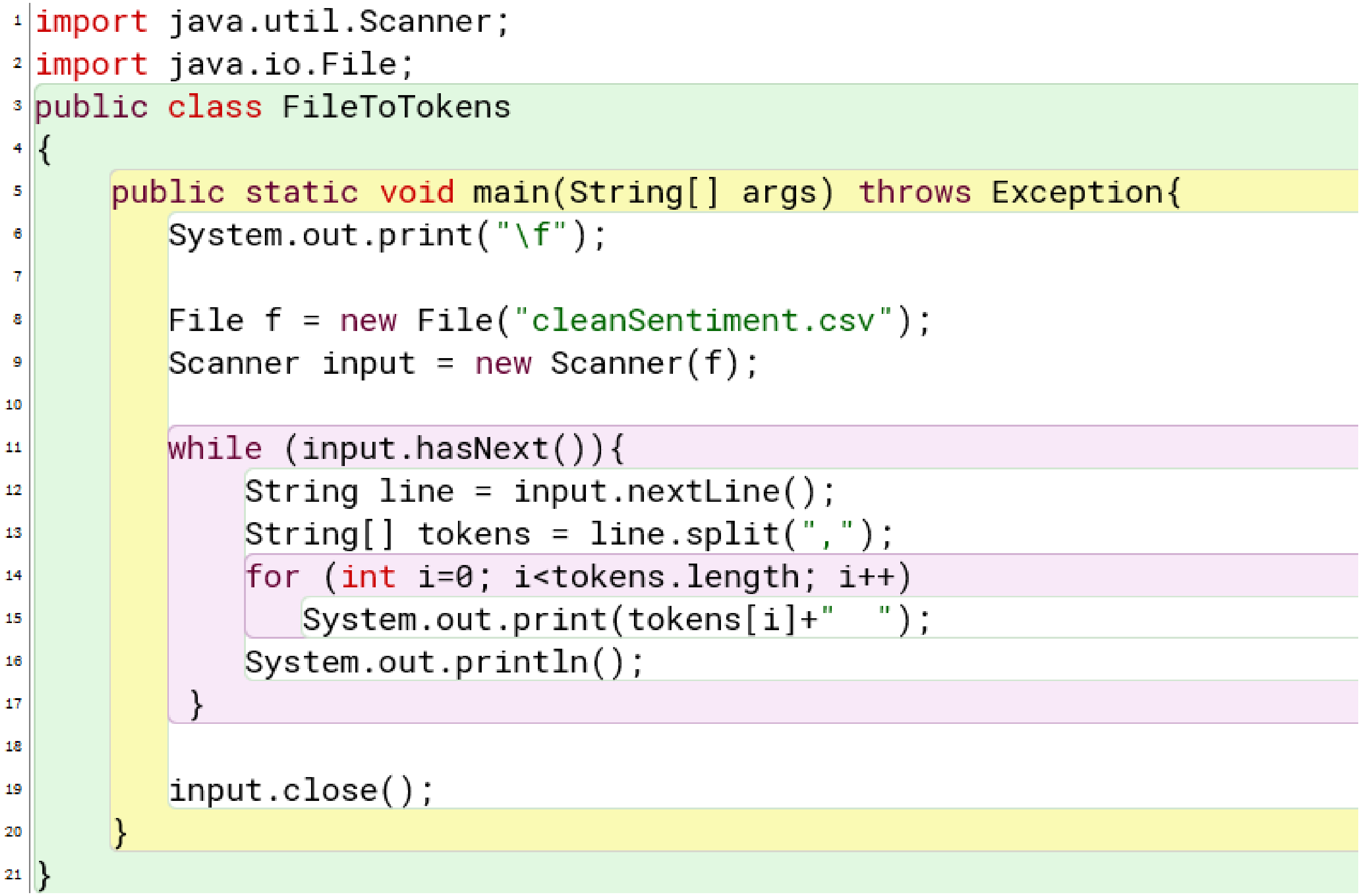
Tokenization

LECTURE 7

Tokenization

- The “cleanSentiment.csv” file is read in as a stream of string lines. However, this stream of string lines is not data.
- The stream needs to be broken down to tokens (words) and then into data structure.

Tokenization By Split(“,”);

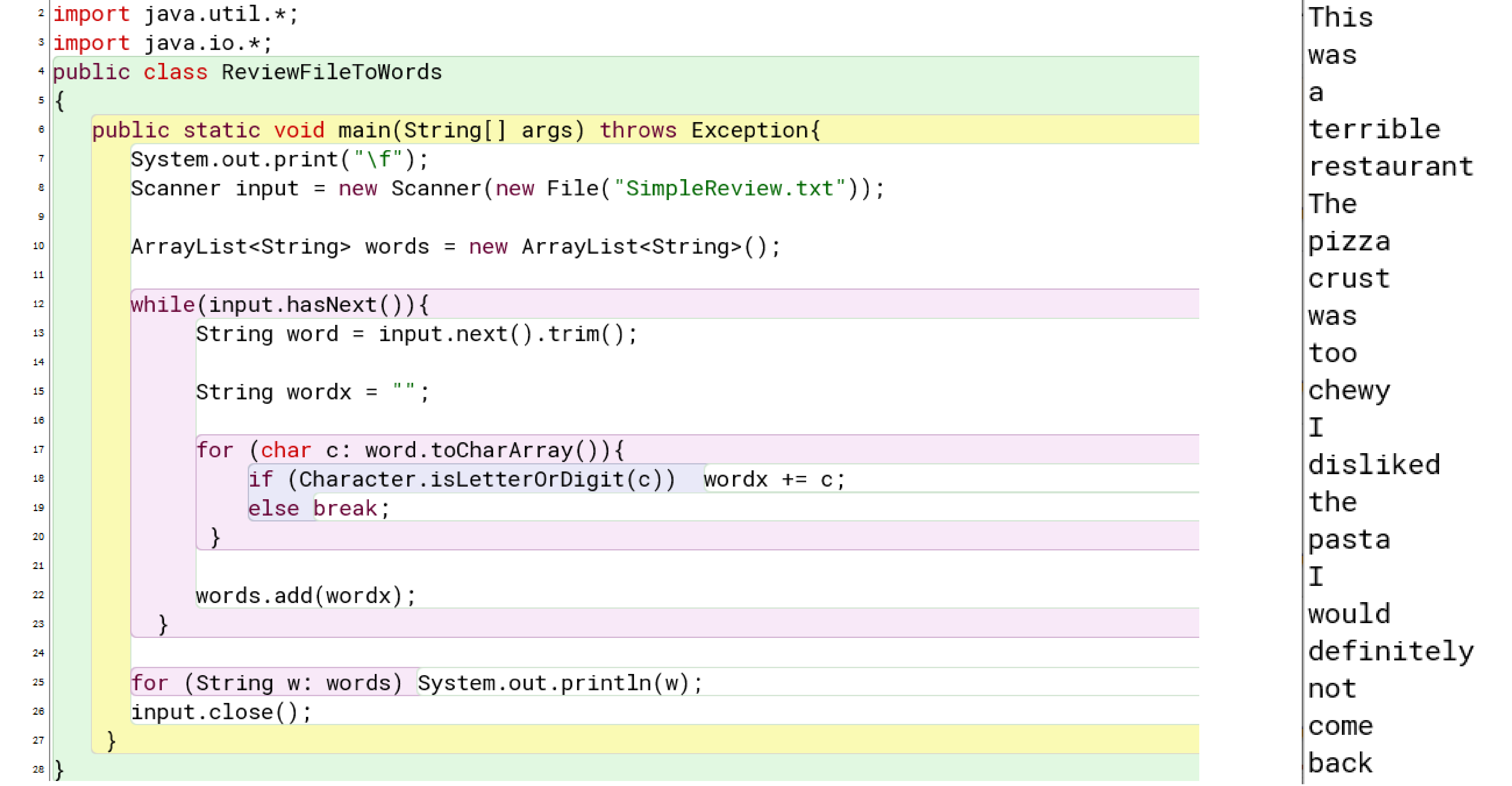


```java
import java.util.Scanner;
import java.io.File;
public class FileToTokens
{
    public static void main(String[] args) throws Exception{
        System.out.print("\f");

        File f = new File("cleanSentiment.csv");
        Scanner input = new Scanner(f);

        while (input.hasNext()){
            String line = input.nextLine();
            String[] tokens = line.split(",");
            for (int i=0; i<tokens.length; i++)
               System.out.print(tokens[i]+"   ");
            System.out.println();
         }

        input.close();
    }
}
```

```
aboard  -0.15
above  0.2
abrupt  -0.83
abruptly  -1.11
abs  0.58
absence  -0.8
absent  0.08
absolute  -1.51
absolutely  0.57
abstract  -0.09
absurd  -1.23
abundant  0.83
academic  0.43
academy  0.61
accent  1.73
accept  0.72
acceptable  -0.12
accepted  0.17
access  0.29
accessible  1.06
accident  -3.03
accidental  -2.07
acclaimed  0.49
```

Review Processing

- Read in a text file and tokenize it into an arraylist of tokens.
- Filtering out punctuation marks and the following letters. ("word's" will be cut to "word")
- Keep numbers.
- Filtering out extra spaces.

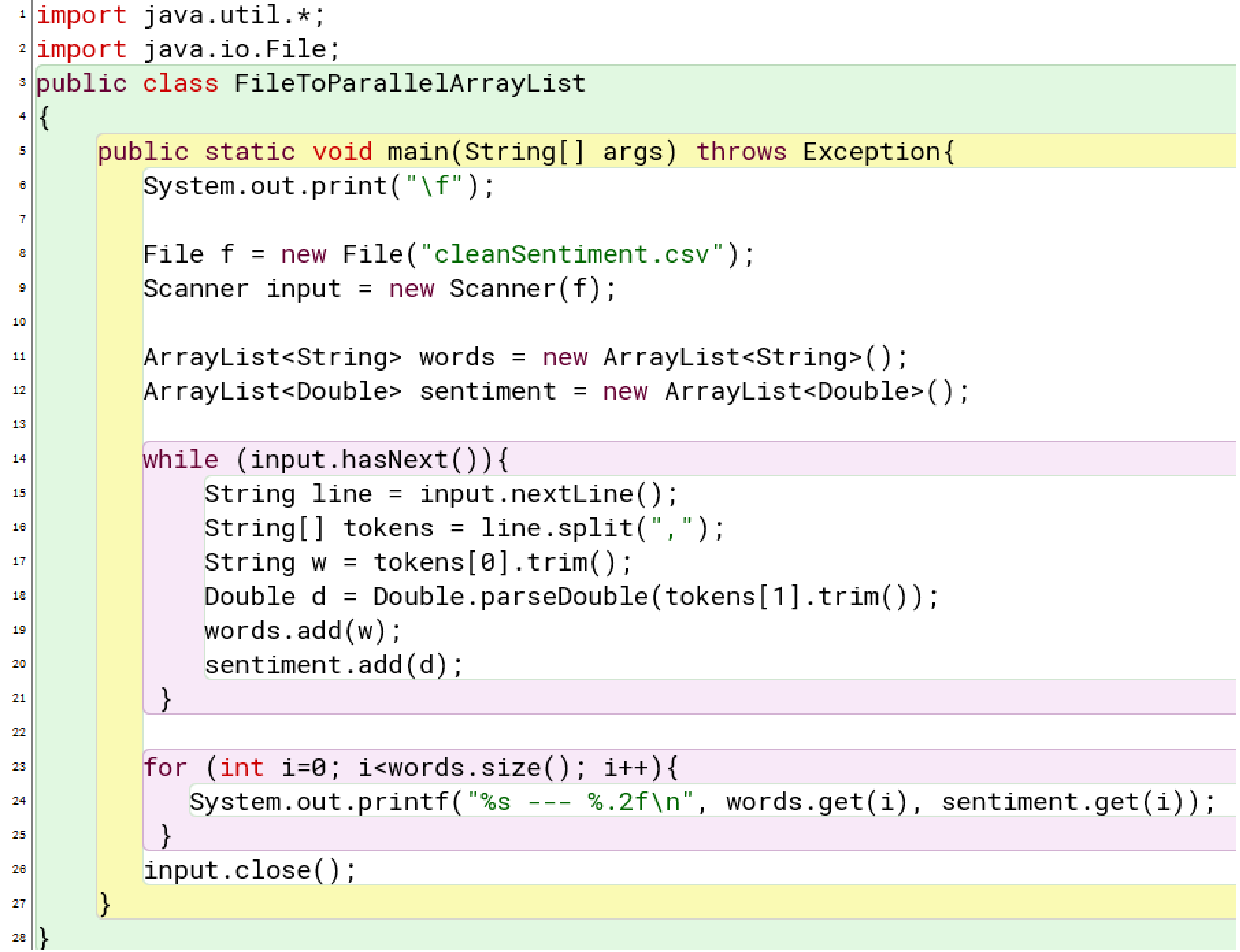


```
import java.util.*;
import java.io.*;
public class ReviewFileToWords
{
    public static void main(String[] args) throws Exception{
        System.out.print("\f");
        Scanner input = new Scanner(new File("SimpleReview.txt"));

        ArrayList<String> words = new ArrayList<String>();

        while(input.hasNext()){
            String word = input.next().trim();

            String wordx = "";

            for (char c: word.toCharArray()){
                if (Character.isLetterOrDigit(c))  wordx += c;
                else break;
             }

            words.add(wordx);
         }

        for (String w: words) System.out.println(w);
        input.close();
    }
}
```

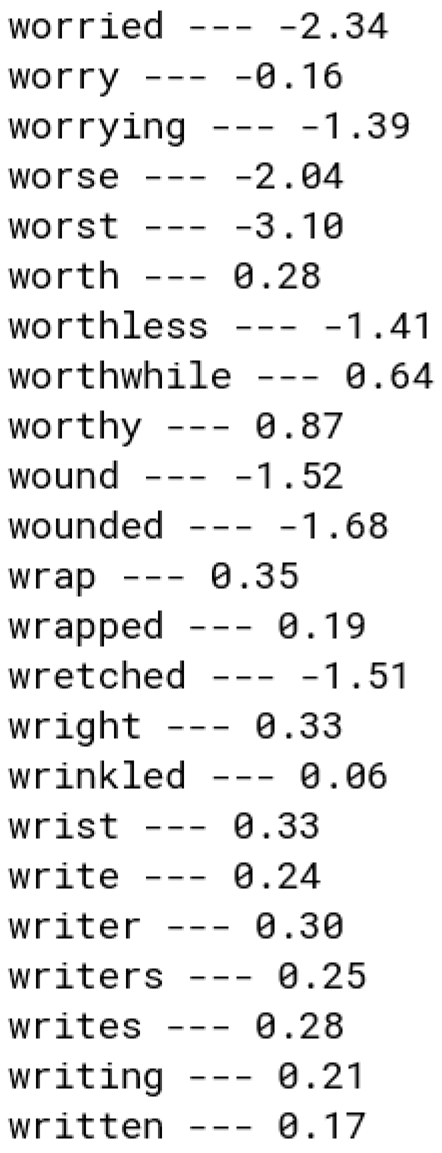
```
This
was
a
terrible
restaurant
The
pizza
crust
was
too
chewy
I
disliked
the
pasta
I
would
definitely
not
come
back
```

Read cleanSentiment to Parallel ArrayList

- Reach each line of the **cleanSentiment** to a parallel arraylist, words and sentiment.



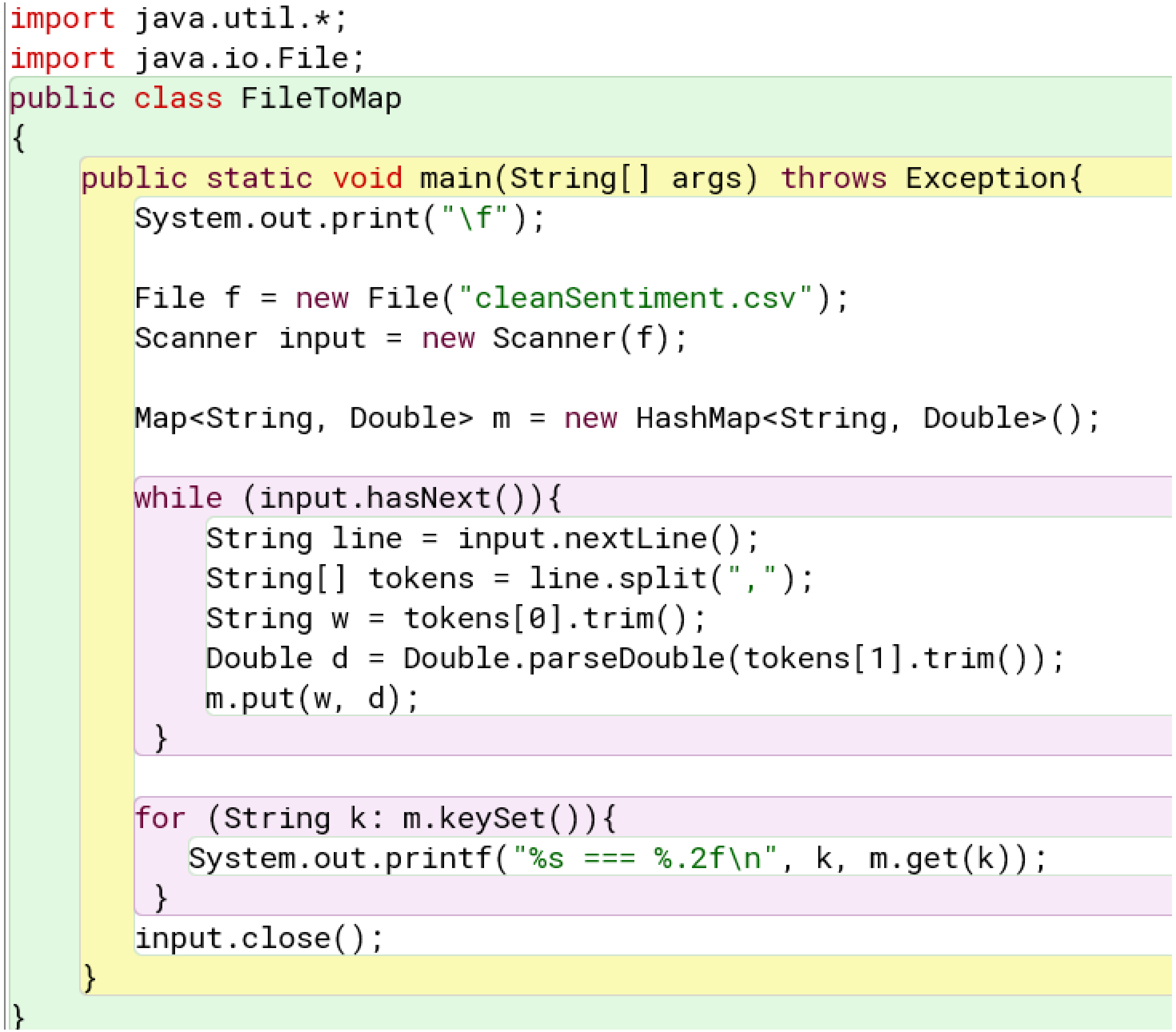
```java
import java.util.*;
import java.io.File;
public class FileToParallelArrayList
{
    public static void main(String[] args) throws Exception{
        System.out.print("\f");

        File f = new File("cleanSentiment.csv");
        Scanner input = new Scanner(f);

        ArrayList<String> words = new ArrayList<String>();
        ArrayList<Double> sentiment = new ArrayList<Double>();

        while (input.hasNext()){
            String line = input.nextLine();
            String[] tokens = line.split(",");
            String w = tokens[0].trim();
            Double d = Double.parseDouble(tokens[1].trim());
            words.add(w);
            sentiment.add(d);
         }

        for (int i=0; i<words.size(); i++){
           System.out.printf("%s --- %.2f\n", words.get(i), sentiment.get(i));
         }
        input.close();
    }
}
```



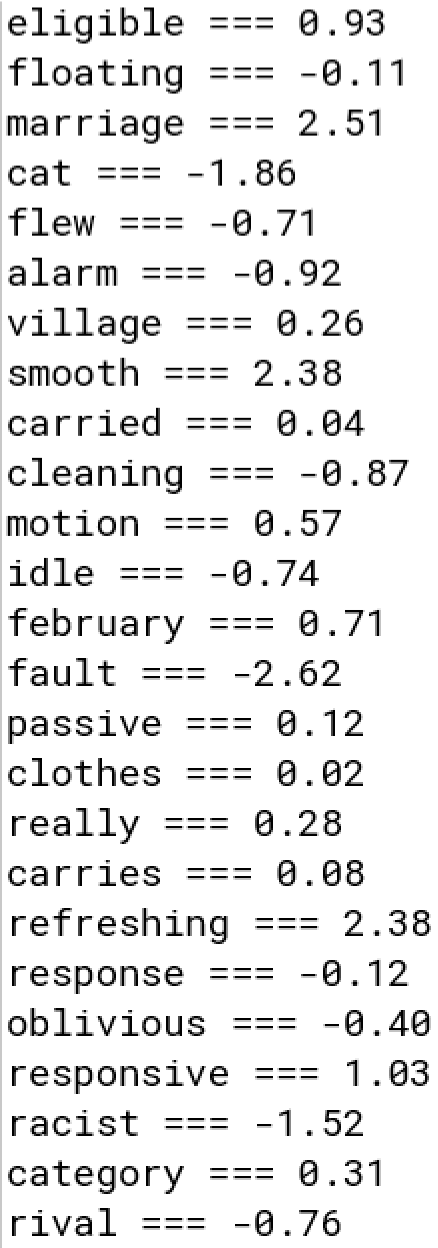
```
worried --- -2.34
worry --- -0.16
worrying --- -1.39
worse --- -2.04
worst --- -3.10
worth --- 0.28
worthless --- -1.41
worthwhile --- 0.64
worthy --- 0.87
wound --- -1.52
wounded --- -1.68
wrap --- 0.35
wrapped --- 0.19
wretched --- -1.51
wright --- 0.33
wrinkled --- 0.06
wrist --- 0.33
write --- 0.24
writer --- 0.30
writers --- 0.25
writes --- 0.28
writing --- 0.21
written --- 0.17
```

Read cleanSentiment to HashMap

- Reach each line of the **cleanSentiment** to a map of words and sentiment.



```java
import java.util.*;
import java.io.File;
public class FileToMap
{
    public static void main(String[] args) throws Exception{
        System.out.print("\f");

        File f = new File("cleanSentiment.csv");
        Scanner input = new Scanner(f);

        Map<String, Double> m = new HashMap<String, Double>();

        while (input.hasNext()){
            String line = input.nextLine();
            String[] tokens = line.split(",");
            String w = tokens[0].trim();
            Double d = Double.parseDouble(tokens[1].trim());
            m.put(w, d);
         }

        for (String k: m.keySet()){
           System.out.printf("%s === %.2f\n", k, m.get(k));
         }
        input.close();
    }
}
```

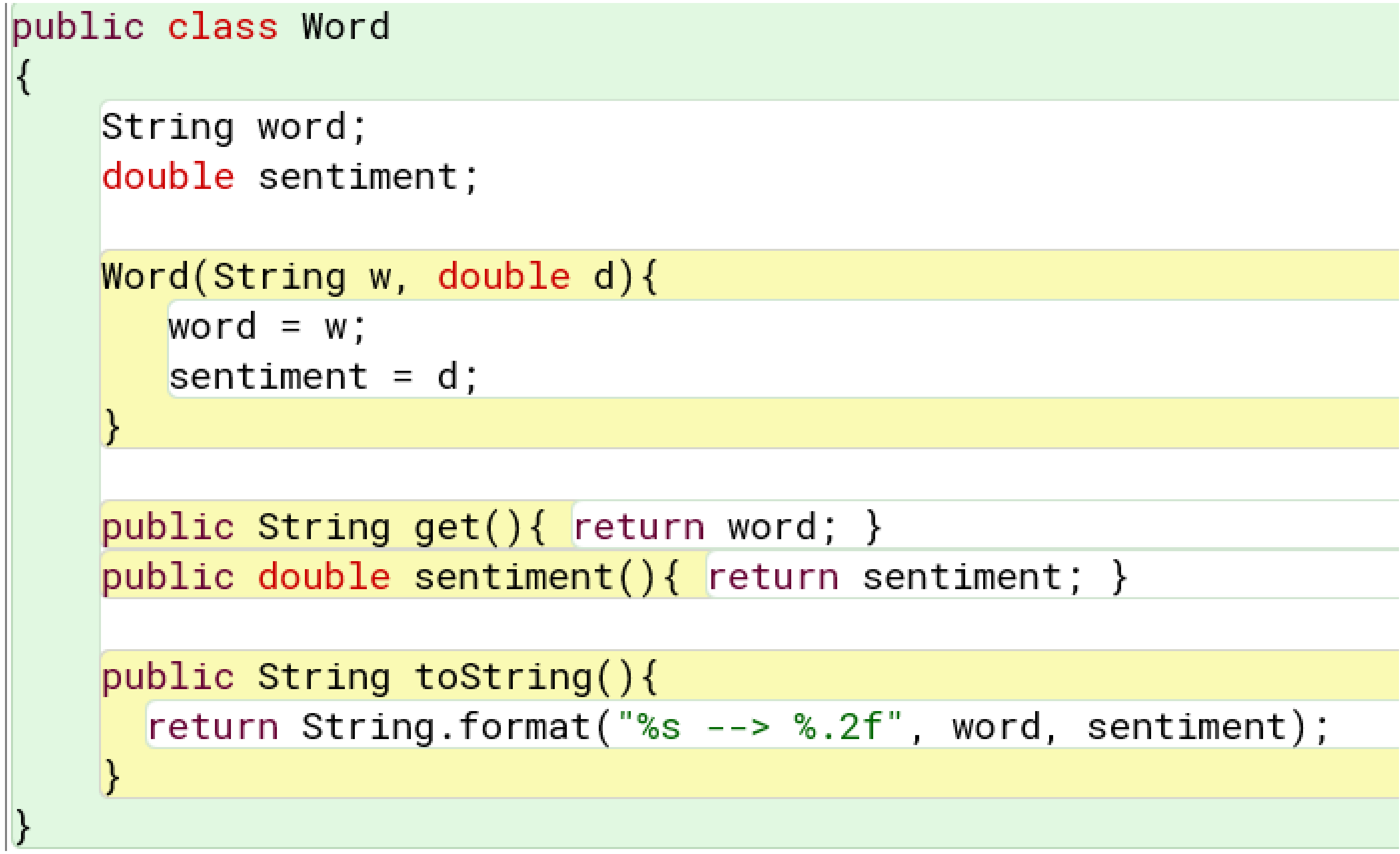


```
eligible === 0.93
floating === -0.11
marriage === 2.51
cat === -1.86
flew === -0.71
alarm === -0.92
village === 0.26
smooth === 2.38
carried === 0.04
cleaning === -0.87
motion === 0.57
idle === -0.74
february === 0.71
fault === -2.62
passive === 0.12
clothes === 0.02
really === 0.28
carries === 0.08
refreshing === 2.38
response === -0.12
oblivious === -0.40
responsive === 1.03
racist === -1.52
category === 0.31
rival === -0.76
```

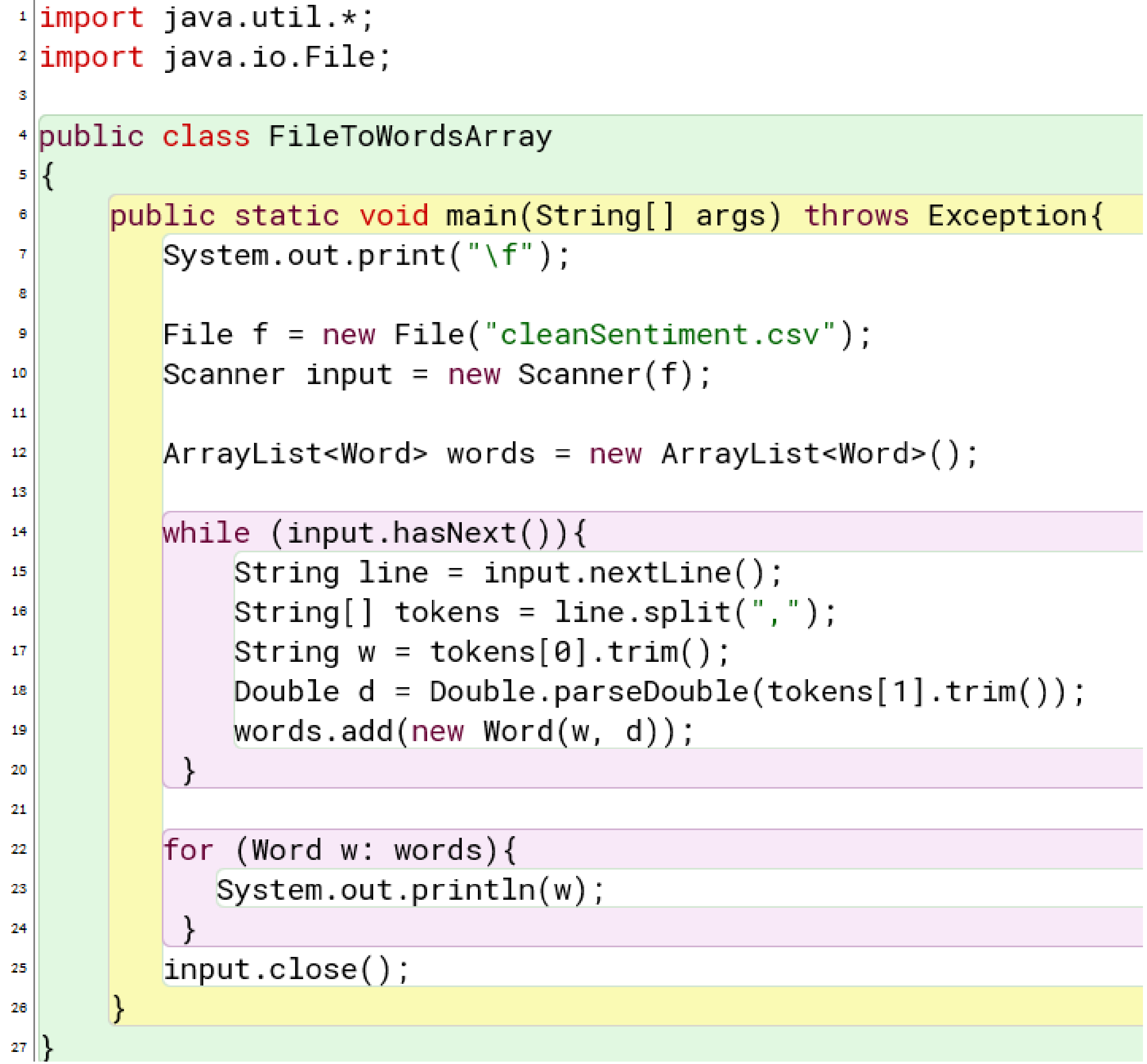
Read cleanSentiment to An ArrayList of Words

- Reach each line of the **cleanSentiment** to an ArrayList of words each of which is an object.

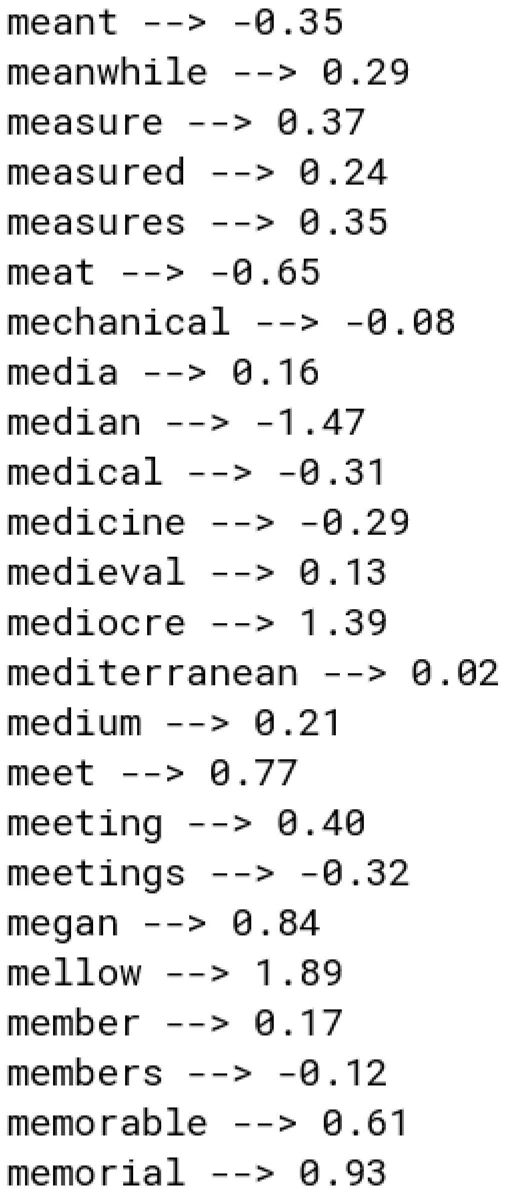
Word Class



```java
public class Word
{
    String word;
    double sentiment;

    Word(String w, double d){
       word = w;
       sentiment = d;
    }

    public String get(){ return word; }
    public double sentiment(){ return sentiment; }

    public String toString(){
      return String.format("%s --> %.2f", word, sentiment);
    }
}
```



```java
import java.util.*;
import java.io.File;

public class FileToWordsArray
{
    public static void main(String[] args) throws Exception{
        System.out.print("\f");

        File f = new File("cleanSentiment.csv");
        Scanner input = new Scanner(f);

        ArrayList<Word> words = new ArrayList<Word>();

        while (input.hasNext()){
            String line = input.nextLine();
            String[] tokens = line.split(",");
            String w = tokens[0].trim();
            Double d = Double.parseDouble(tokens[1].trim());
            words.add(new Word(w, d));
        }

        for (Word w: words){
           System.out.println(w);
        }
        input.close();
    }
}
```



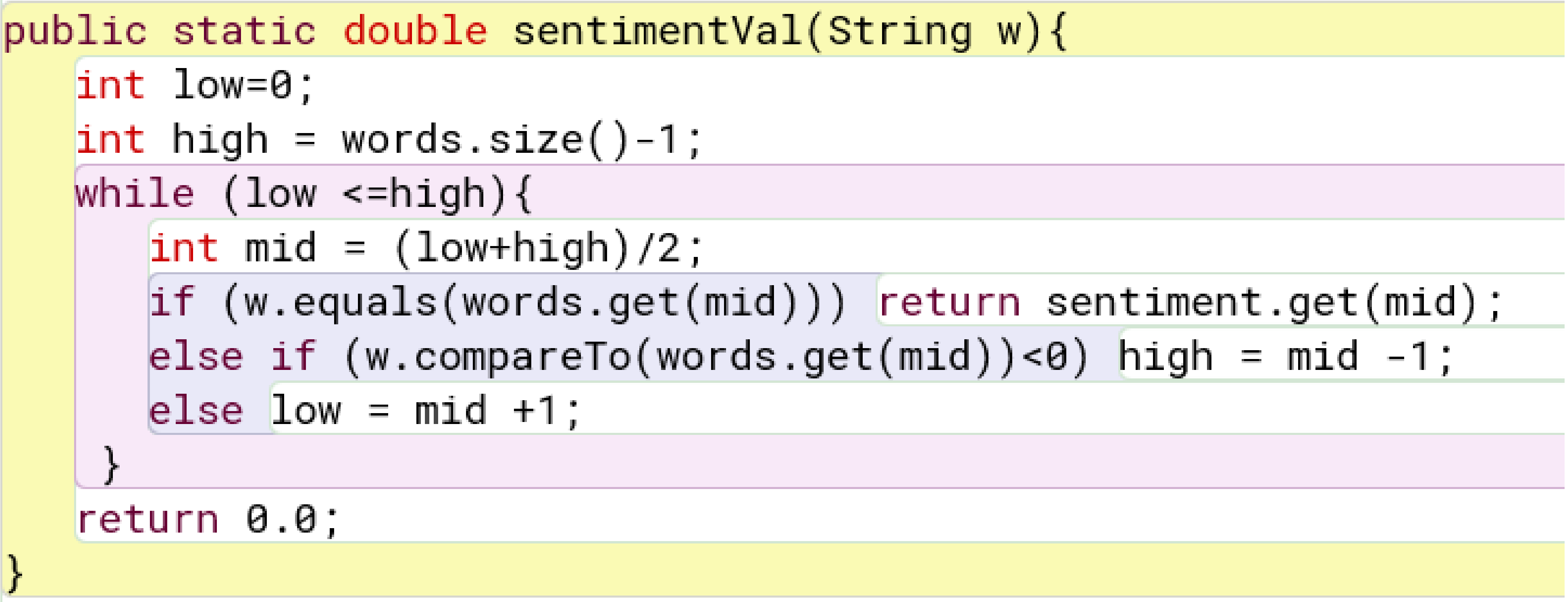
```
meant --> -0.35
meanwhile --> 0.29
measure --> 0.37
measured --> 0.24
measures --> 0.35
meat --> -0.65
mechanical --> -0.08
media --> 0.16
median --> -1.47
medical --> -0.31
medicine --> -0.29
medieval --> 0.13
mediocre --> 1.39
mediterranean --> 0.02
medium --> 0.21
meet --> 0.77
meeting --> 0.40
meetings --> -0.32
megan --> 0.84
mellow --> 1.89
member --> 0.17
members --> -0.12
memorable --> 0.61
memorial --> 0.93
```

Sentimental Analysis

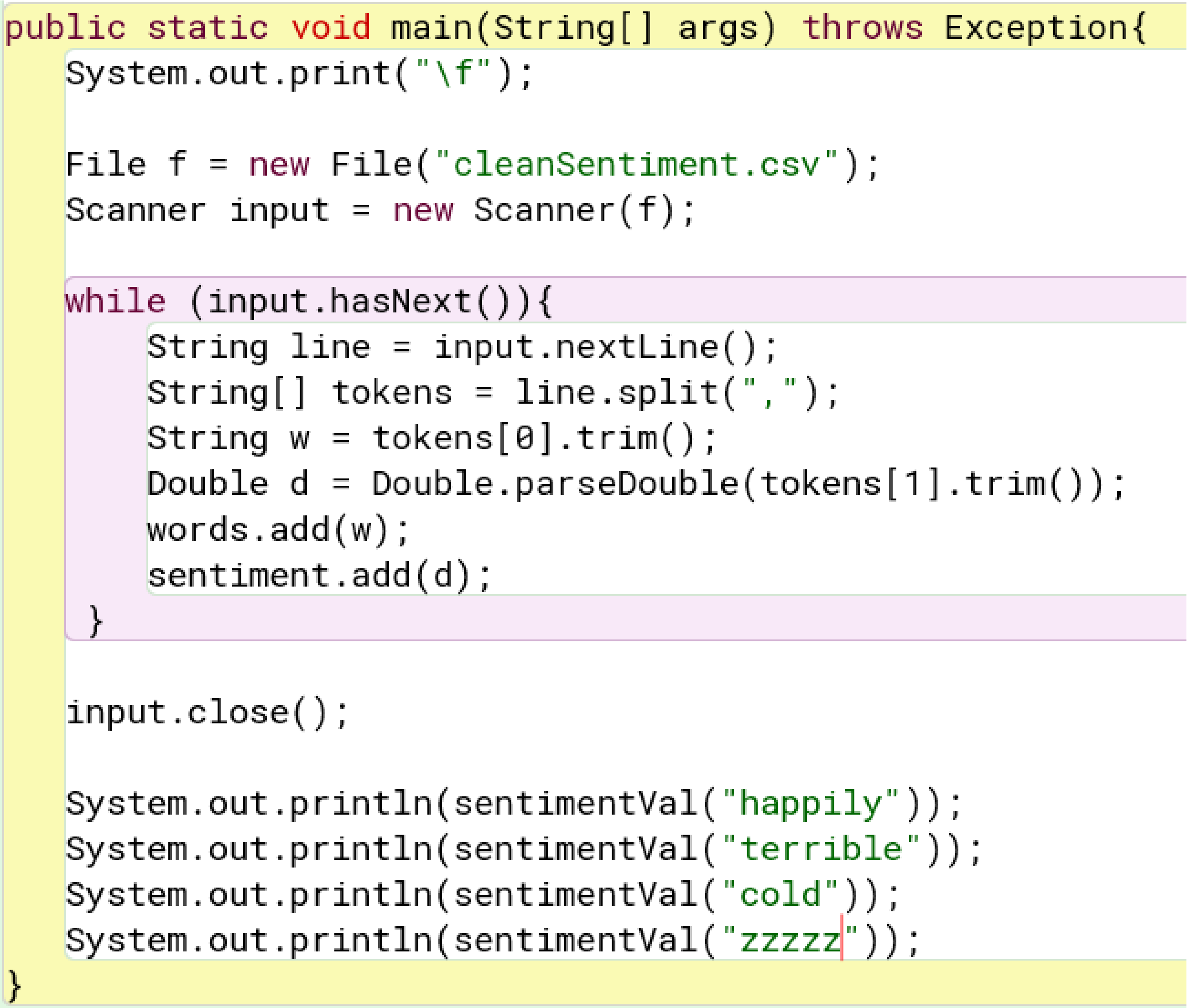
LECTURE 8

sentimentVal(String word)

- Return 0 if not found.
- Return the sentiment value if the word exists in the word list.
- Both parallel array and ArrayList<Word> need to use binary search.

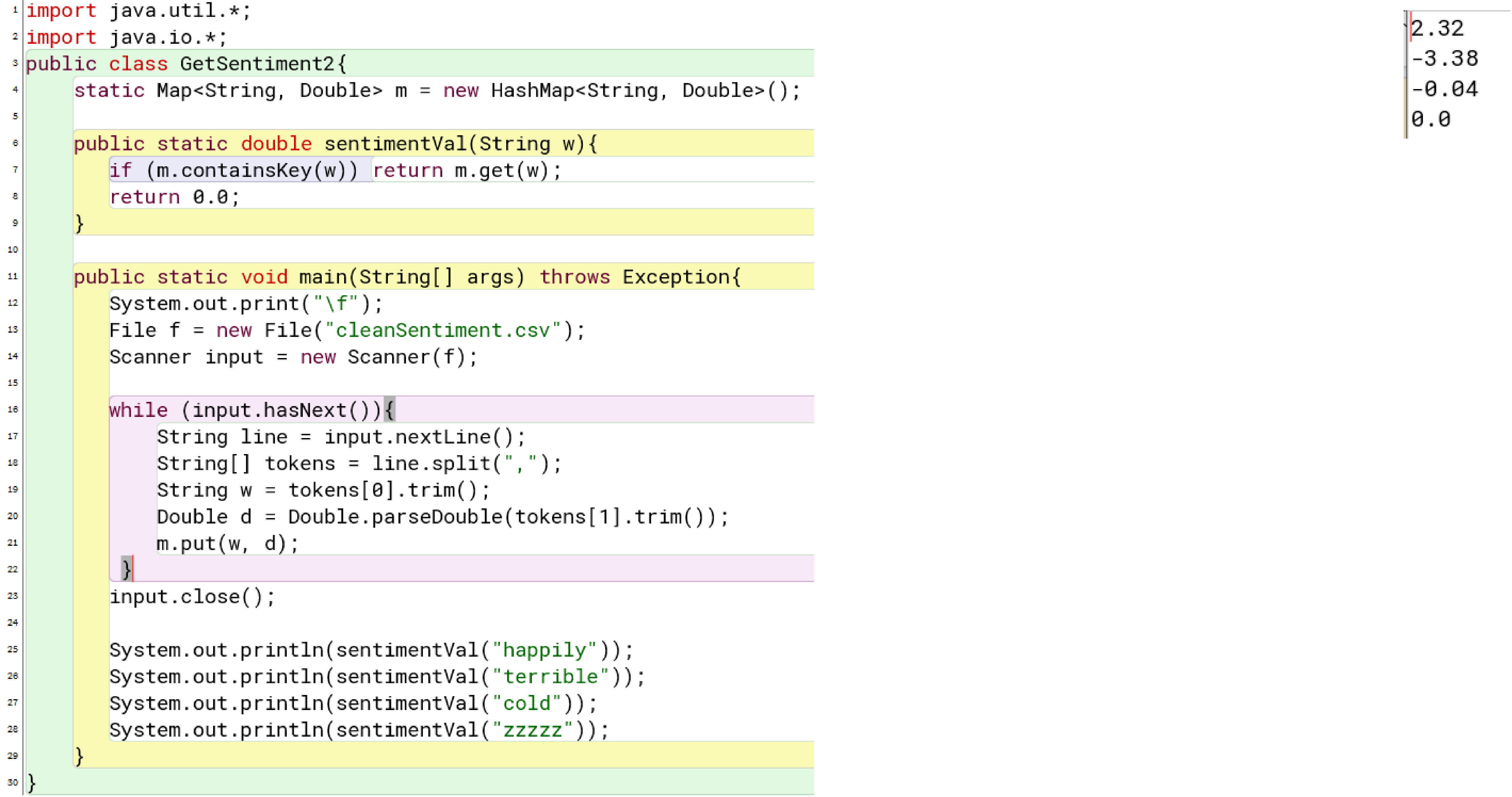


```
public static double sentimentVal(String w){
    int low=0;
    int high = words.size()-1;
    while (low <=high){
        int mid = (low+high)/2;
        if (w.equals(words.get(mid))) return sentiment.get(mid);
        else if (w.compareTo(words.get(mid))<0) high = mid -1;
        else low = mid +1;
     }
    return 0.0;
}
```



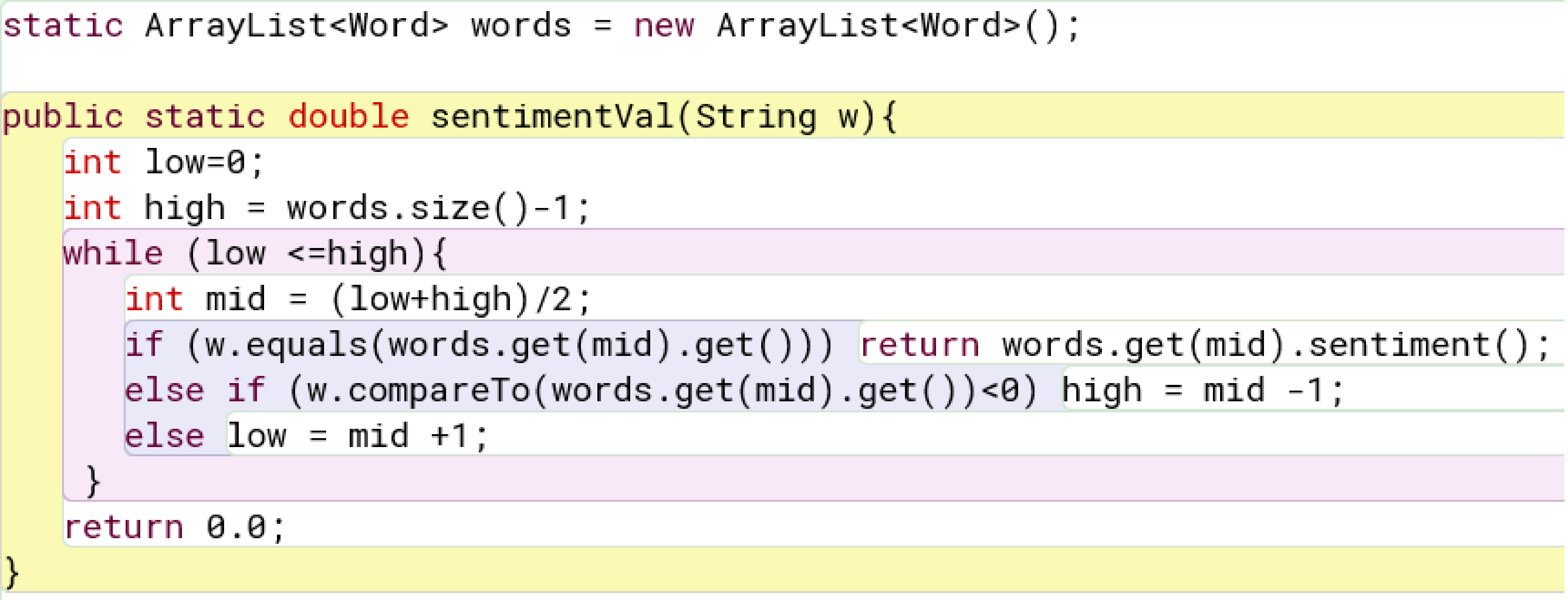
```java
public static void main(String[] args) throws Exception{
    System.out.print("\f");

    File f = new File("cleanSentiment.csv");
    Scanner input = new Scanner(f);

    while (input.hasNext()){
        String line = input.nextLine();
        String[] tokens = line.split(",");
        String w = tokens[0].trim();
        Double d = Double.parseDouble(tokens[1].trim());
        words.add(w);
        sentiment.add(d);
     }

    input.close();

    System.out.println(sentimentVal("happily"));
    System.out.println(sentimentVal("terrible"));
    System.out.println(sentimentVal("cold"));
    System.out.println(sentimentVal("zzzzz"));
}
```

```
2.32
-3.38
-0.04
0.0
```

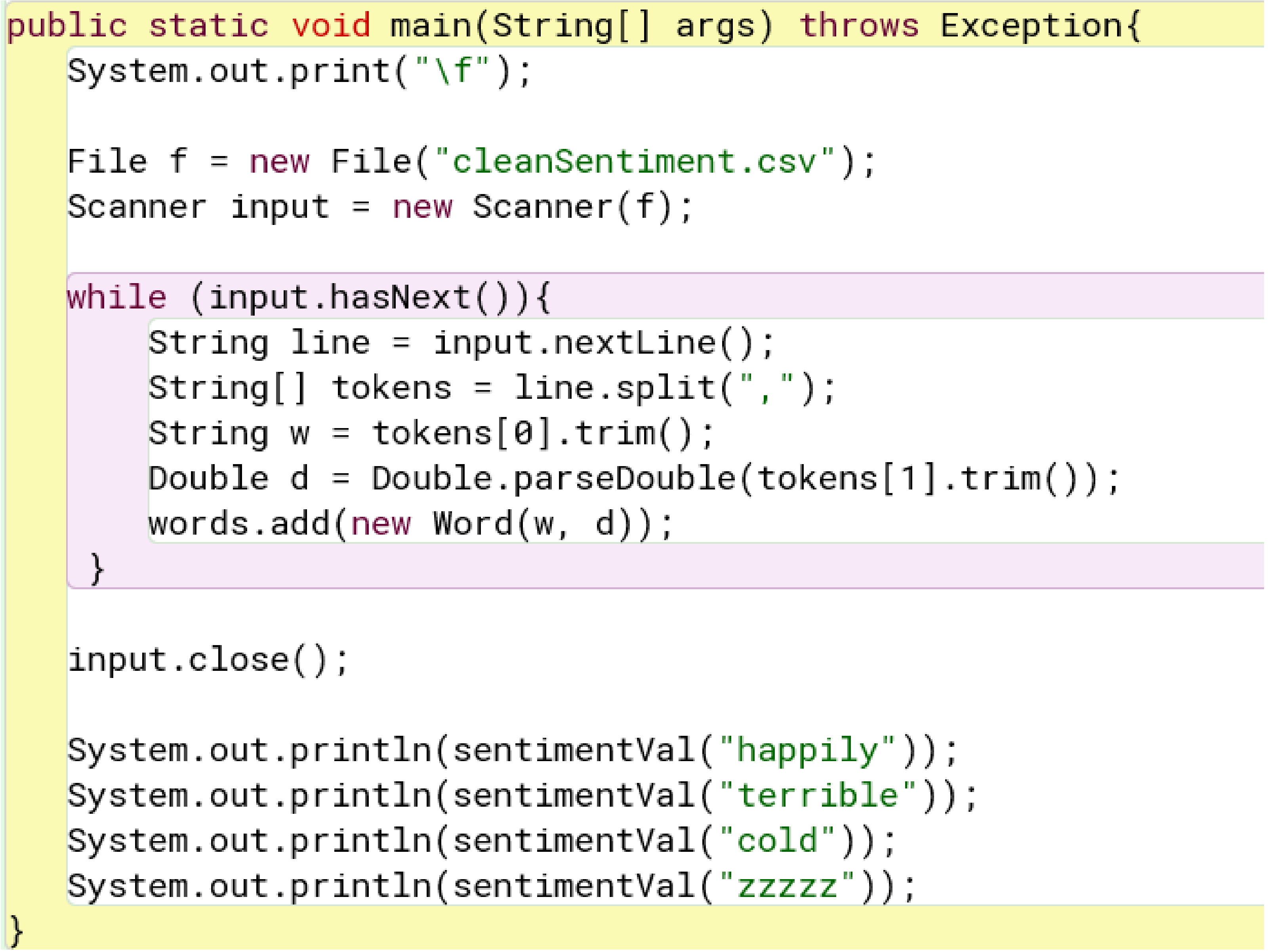


```java
import java.util.*;
import java.io.*;
public class GetSentiment2{
    static Map<String, Double> m = new HashMap<String, Double>();

    public static double sentimentVal(String w){
        if (m.containsKey(w)) return m.get(w);
        return 0.0;
    }

    public static void main(String[] args) throws Exception{
        System.out.print("\f");
        File f = new File("cleanSentiment.csv");
        Scanner input = new Scanner(f);

        while (input.hasNext()){
            String line = input.nextLine();
            String[] tokens = line.split(",");
            String w = tokens[0].trim();
            Double d = Double.parseDouble(tokens[1].trim());
            m.put(w, d);
         }
        input.close();

        System.out.println(sentimentVal("happily"));
        System.out.println(sentimentVal("terrible"));
        System.out.println(sentimentVal("cold"));
        System.out.println(sentimentVal("zzzzz"));
    }
}
```

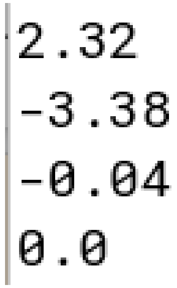
```
2.32
-3.38
-0.04
0.0
```



```java
static ArrayList<Word> words = new ArrayList<Word>();

public static double sentimentVal(String w){
    int low=0;
    int high = words.size()-1;
    while (low <=high){
        int mid = (low+high)/2;
        if (w.equals(words.get(mid).get())) return words.get(mid).sentiment();
        else if (w.compareTo(words.get(mid).get())<0) high = mid -1;
        else low = mid +1;
     }
    return 0.0;
}
```



```java
public static void main(String[] args) throws Exception{
    System.out.print("\f");

    File f = new File("cleanSentiment.csv");
    Scanner input = new Scanner(f);

    while (input.hasNext()){
        String line = input.nextLine();
        String[] tokens = line.split(",");
        String w = tokens[0].trim();
        Double d = Double.parseDouble(tokens[1].trim());
        words.add(new Word(w, d));
     }

    input.close();

    System.out.println(sentimentVal("happily"));
    System.out.println(sentimentVal("terrible"));
    System.out.println(sentimentVal("cold"));
    System.out.println(sentimentVal("zzzzz"));
}
```



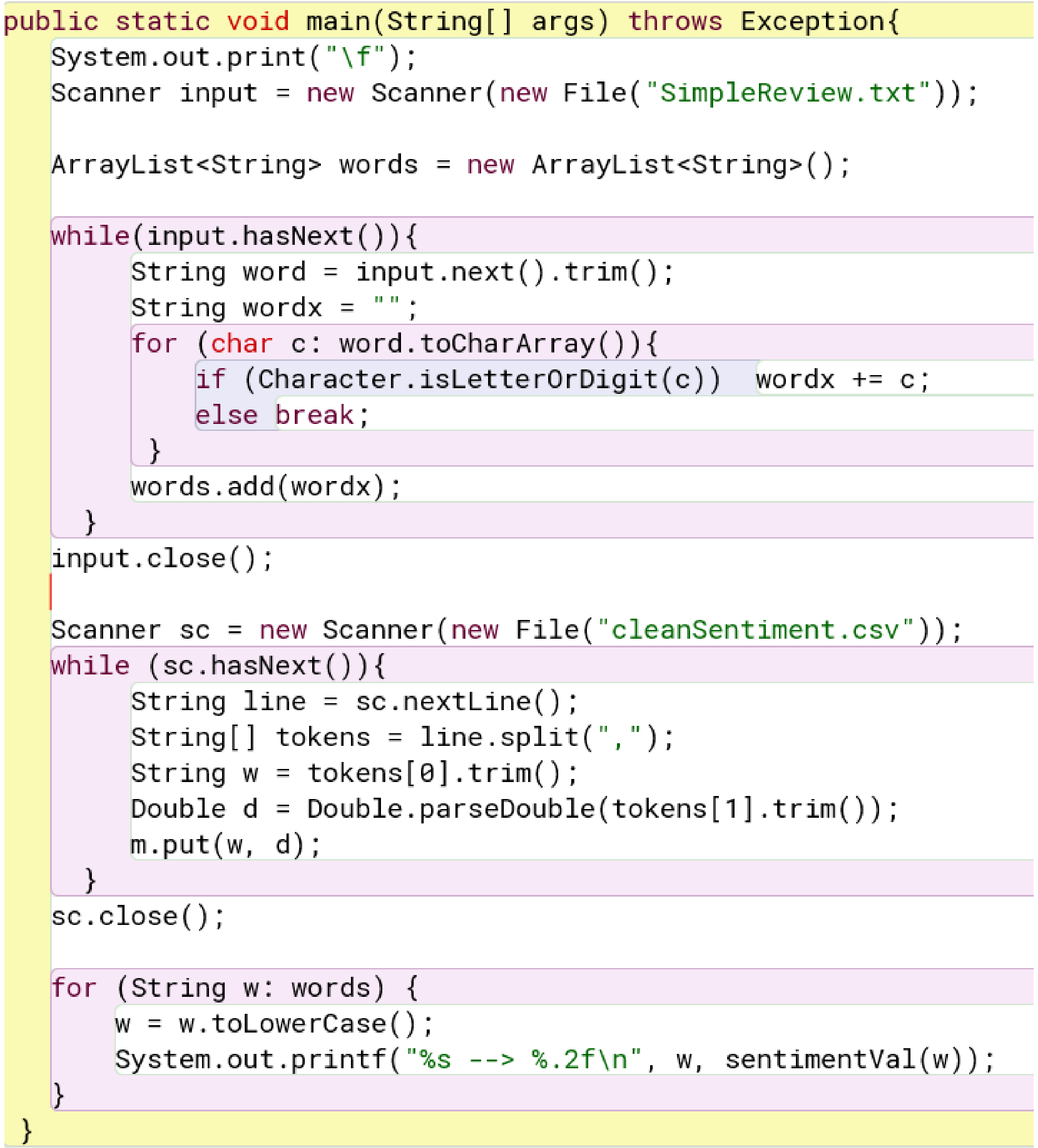
```
2.32
-3.38
-0.04
0.0
```

sentimentVal(String word)

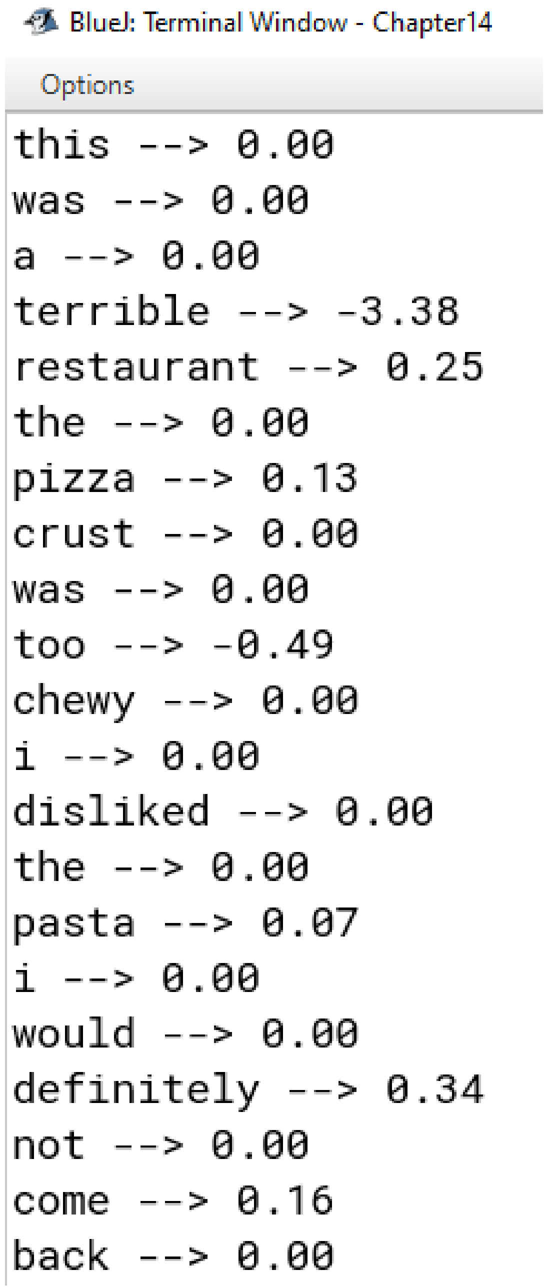
- The version using map is the most efficient one. And, in terms of coding, it is also the easiest one.

Sentimental Analysis

- Go through every word in the “SimpleReview.txt”, get its sentimentVal(word) for each word.



```java
public static void main(String[] args) throws Exception{
    System.out.print("\f");
    Scanner input = new Scanner(new File("SimpleReview.txt"));

    ArrayList<String> words = new ArrayList<String>();

    while(input.hasNext()){
        String word = input.next().trim();
        String wordx = "";
        for (char c: word.toCharArray()){
            if (Character.isLetterOrDigit(c))  wordx += c;
            else break;
         }
        words.add(wordx);
    }
    input.close();

    Scanner sc = new Scanner(new File("cleanSentiment.csv"));
    while (sc.hasNext()){
        String line = sc.nextLine();
        String[] tokens = line.split(",");
        String w = tokens[0].trim();
        Double d = Double.parseDouble(tokens[1].trim());
        m.put(w, d);
    }
    sc.close();

    for (String w: words) {
       w = w.toLowerCase();
       System.out.printf("%s --> %.2f\n", w, sentimentVal(w));
    }
 }
```



```
this --> 0.00
was --> 0.00
a --> 0.00
terrible --> -3.38
restaurant --> 0.25
the --> 0.00
pizza --> 0.13
crust --> 0.00
was --> 0.00
too --> -0.49
chewy --> 0.00
i --> 0.00
disliked --> 0.00
the --> 0.00
pasta --> 0.07
i --> 0.00
would --> 0.00
definitely --> 0.34
not --> 0.00
come --> 0.16
back --> 0.00
```

TotalSentimentVal and Star Ratings

LECTURE 9

TotalSentimentVal

- Go through every word in the input file and then calculate the sum of the sentiment value for each word.

Star Rating

- Based on the total sentiment value for each word. Calculate the star rating for the whole costumer’s review.

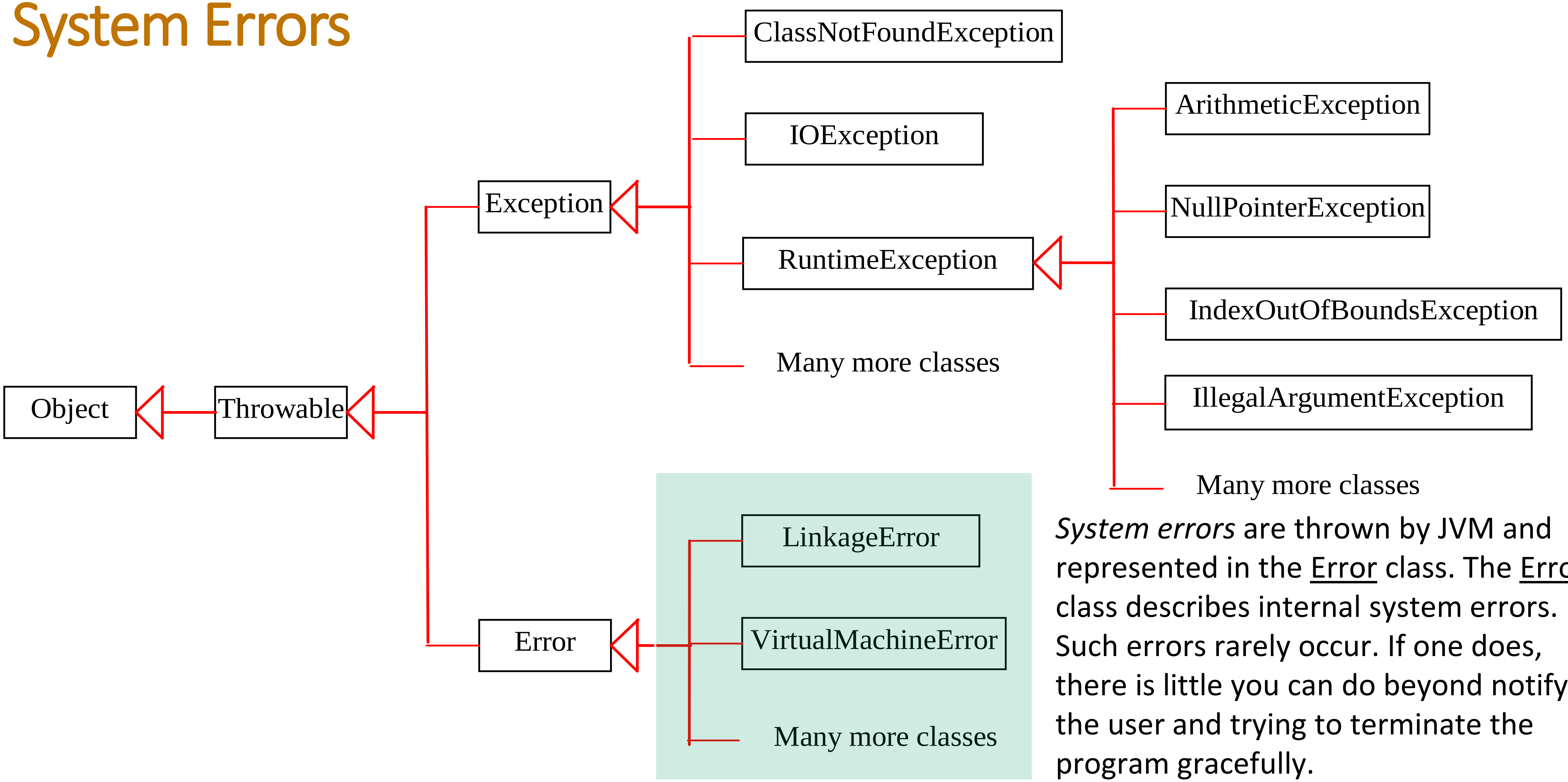
Demonstration Program

TOTALSENTIMENTANALYSIS.JAVA

Exception Types

LECTURE 10

System Errors



System errors are thrown by JVM and represented in the Error class. The Error class describes internal system errors. Such errors rarely occur. If one does, there is little you can do beyond notifying the user and trying to terminate the program gracefully.

System Error Classes

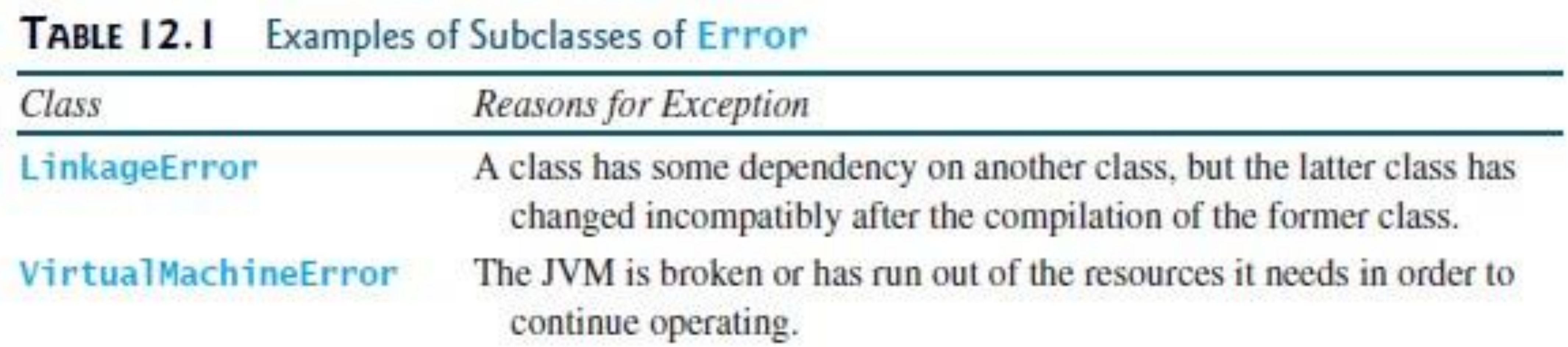
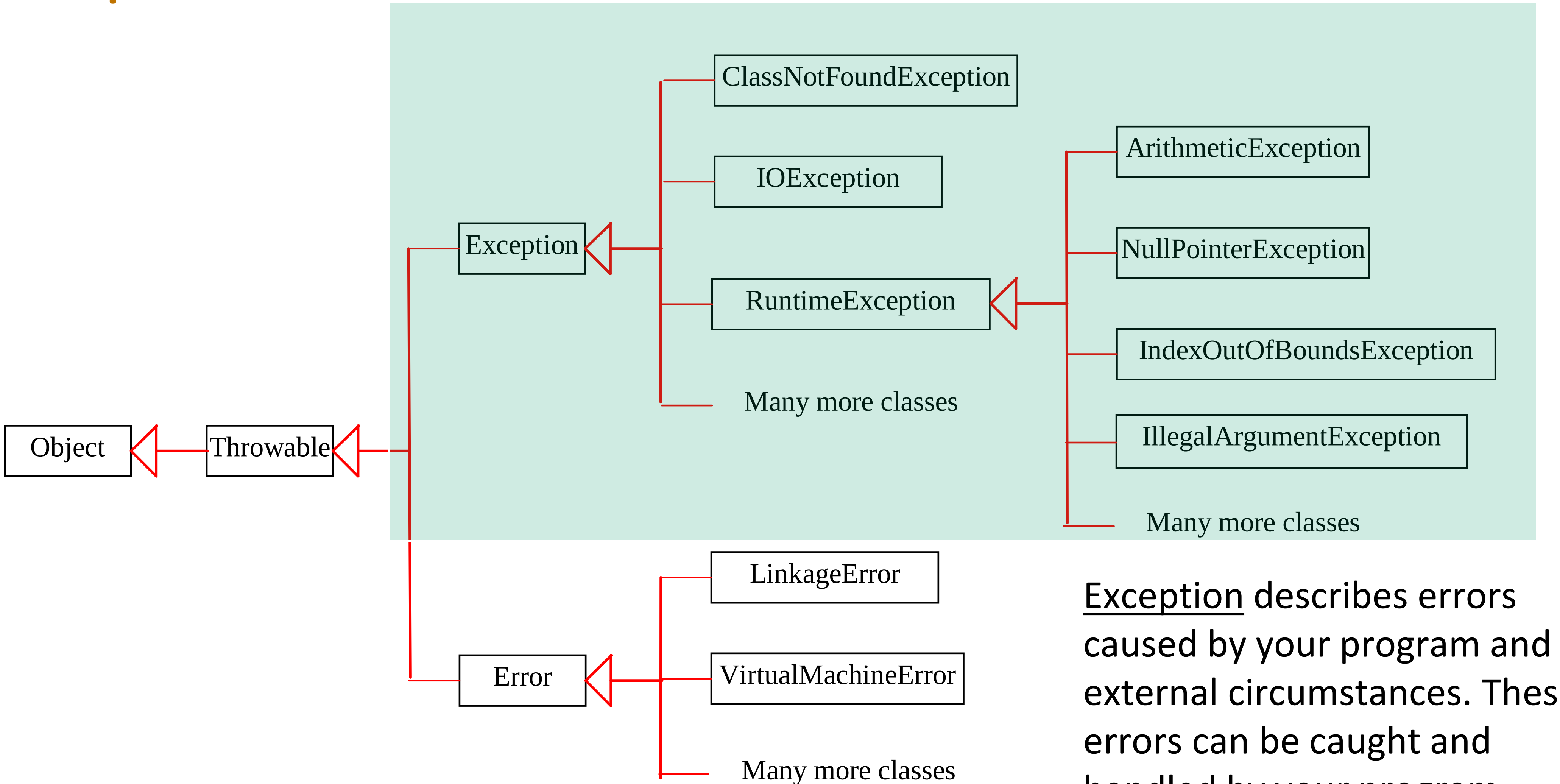


TABLE 12.1 Examples of Subclasses of `Error`

Class	*Reasons for Exception*
`LinkageError`	A class has some dependency on another class, but the latter class has changed incompatibly after the compilation of the former class.
`VirtualMachineError`	The JVM is broken or has run out of the resources it needs in order to continue operating.

Exceptions



Exception describes errors caused by your program and external circumstances. These errors can be caught and handled by your program.

Exception

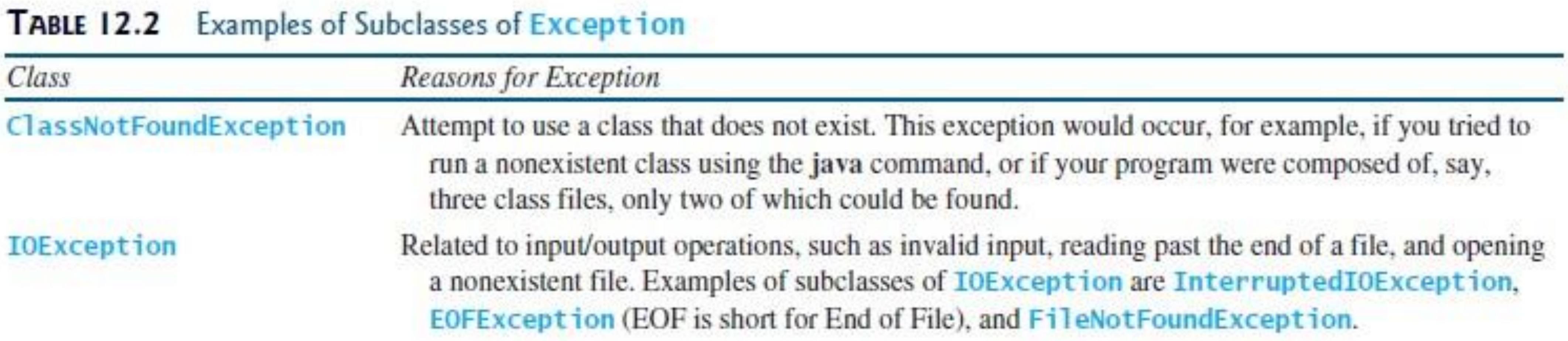
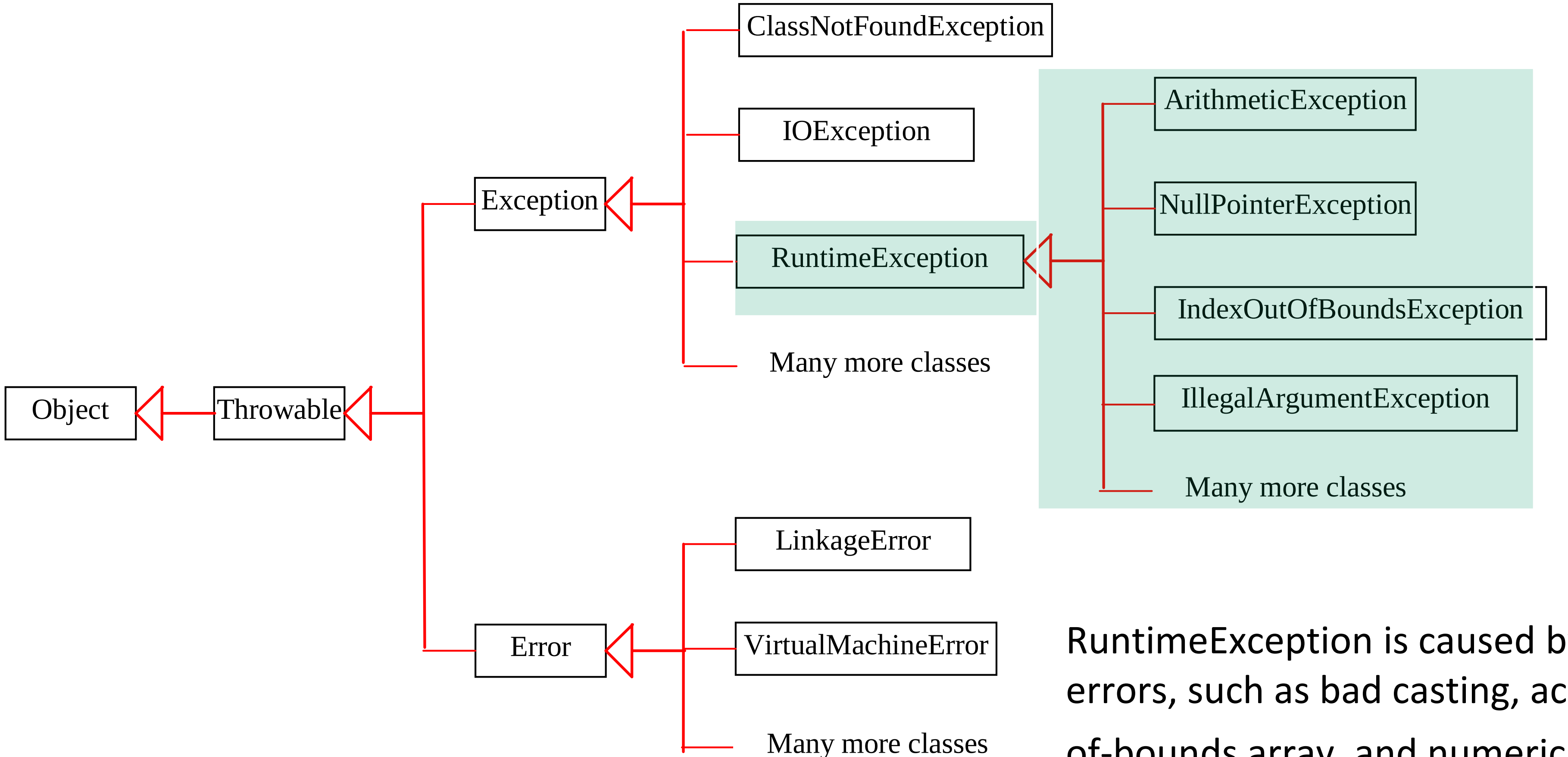


TABLE 12.2 Examples of Subclasses of `Exception`

Class	*Reasons for Exception*
`ClassNotFoundException`	Attempt to use a class that does not exist. This exception would occur, for example, if you tried to run a nonexistent class using the **java** command, or if your program were composed of, say, three class files, only two of which could be found.
`IOException`	Related to input/output operations, such as invalid input, reading past the end of a file, and opening a nonexistent file. Examples of subclasses of `IOException` are `InterruptedIOException`, `EOFException` (EOF is short for End of File), and `FileNotFoundException`.

Runtime Exceptions



RuntimeException is caused by programming errors, such as bad casting, accessing an out-of-bounds array, and numeric errors.

RuntimeException

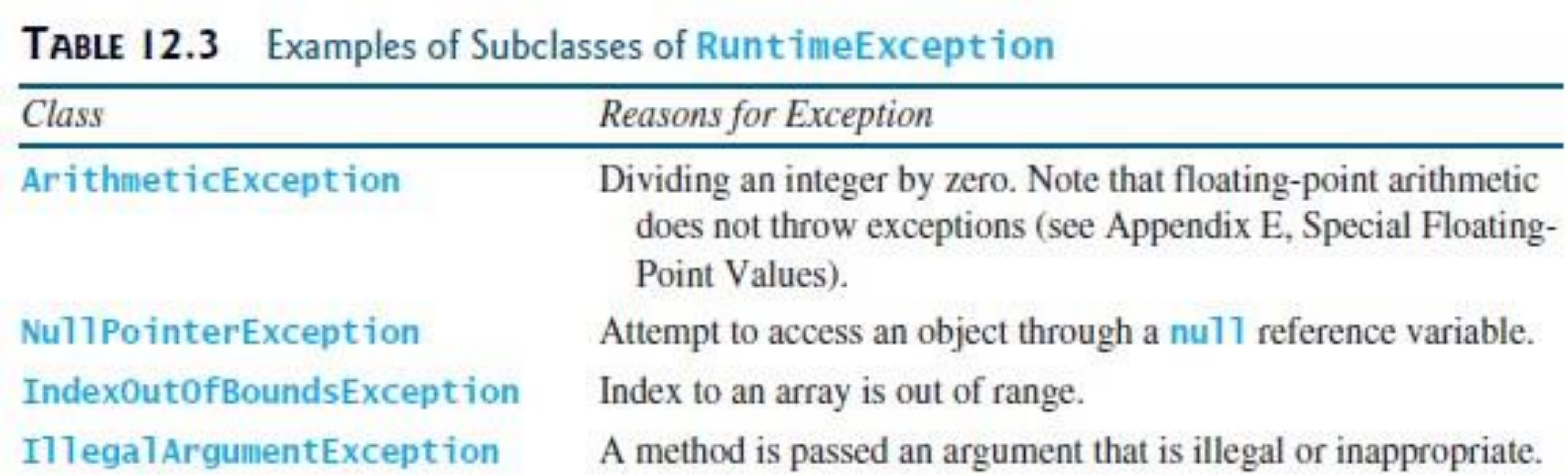


TABLE 12.3 Examples of Subclasses of `RuntimeException`

Class	*Reasons for Exception*
`ArithmeticException`	Dividing an integer by zero. Note that floating-point arithmetic does not throw exceptions (see Appendix E, Special Floating-Point Values).
`NullPointerException`	Attempt to access an object through a `null` reference variable.
`IndexOutOfBoundsException`	Index to an array is out of range.
`IllegalArgumentException`	A method is passed an argument that is illegal or inappropriate.

Checked Exceptions vs. Unchecked Exceptions

- **RuntimeException**, **Error** and their subclasses are known as *unchecked exceptions*. (Compiler won't force you to check it.)
- All other exceptions are known as *checked exceptions*, meaning that the compiler forces the programmer to check and deal with the exceptions. (Such as IOException, if you do nothing about it, compiler won't let you pass.)

Unchecked Exceptions

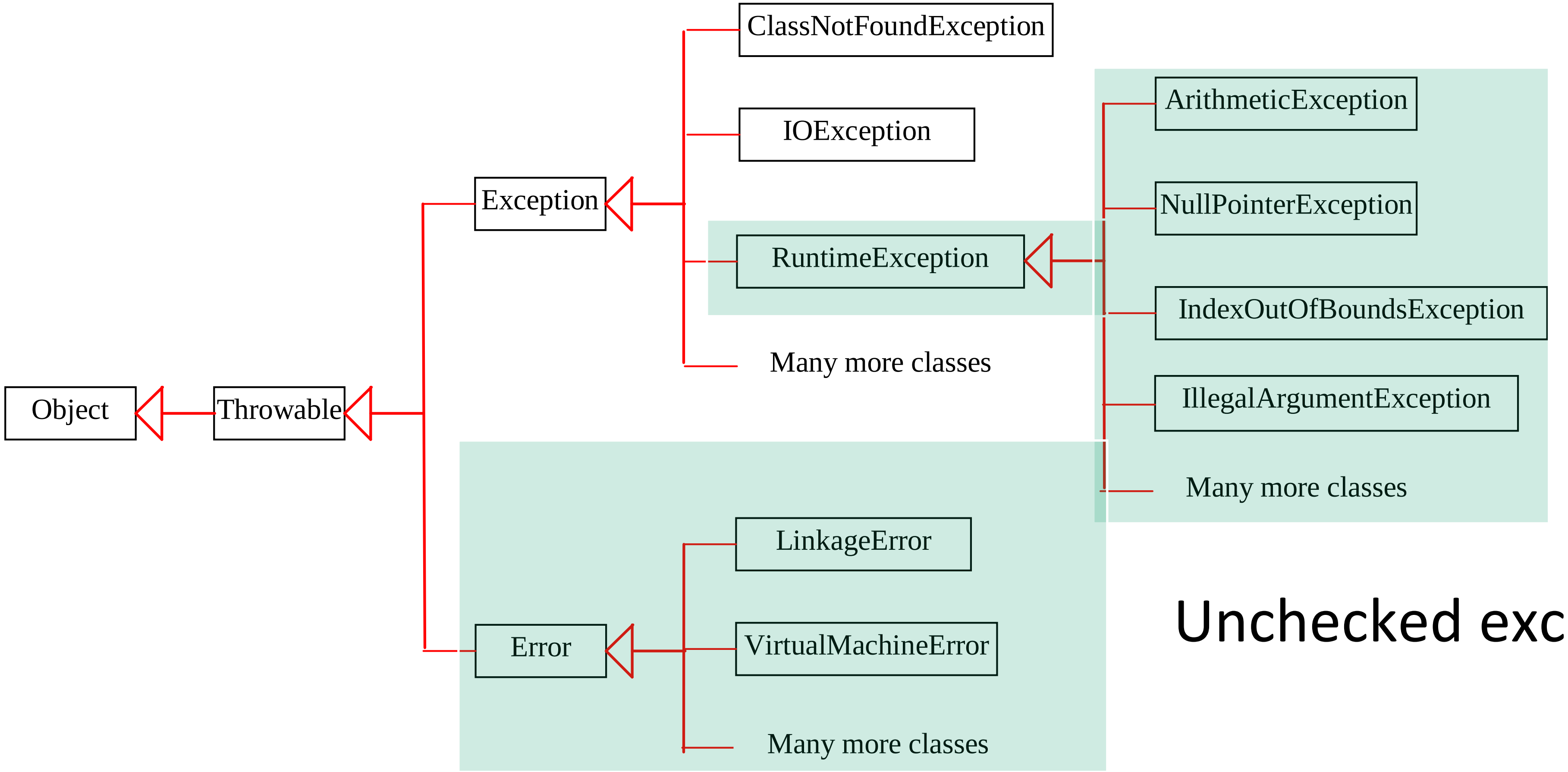
- In most cases, unchecked exceptions reflect programming logic errors that are not recoverable.

- For example, a **NullPointerException** is thrown if you access an object through a reference variable before an object is assigned to it; an **IndexOutOfBoundsException** is thrown if you access an element in an array outside the bounds of the array.

Unchecked Exceptions

- These are the logic errors that should be corrected in the program. Unchecked exceptions can occur anywhere in the program.
- To avoid cumbersome overuse of try-catch blocks, Java does not mandate you to write code to catch unchecked exceptions.

Unchecked Exceptions



Unchecked exception.

Exception Handling

LECTURE 11

Exception & Call Stack

- When an exception occurs inside a Java method, the method creates an **Exception** object and passes the **Exception** object to the **JVM** (in Java term, the method "**throw**" an Exception).
- The Exception object contains the type of the exception, and the state of the program when the exception occurs.

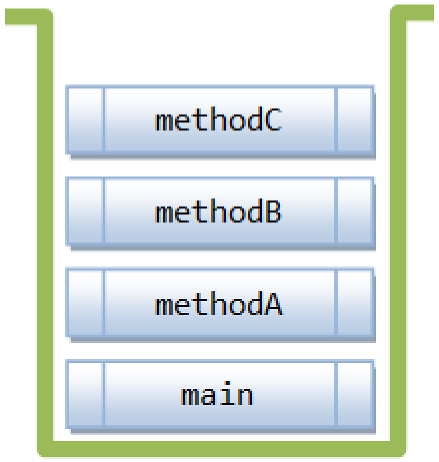
Exception & Call Stack

- The JVM is responsible for finding an exception handler to process the Exception object. It searches backward through the call stack until it finds a matching exception handler for that particular class of Exception object (in Java term, it is called "catch" the Exception).
- **If the JVM cannot find a matching exception handler in all the methods in the call stack, it terminates the program.**

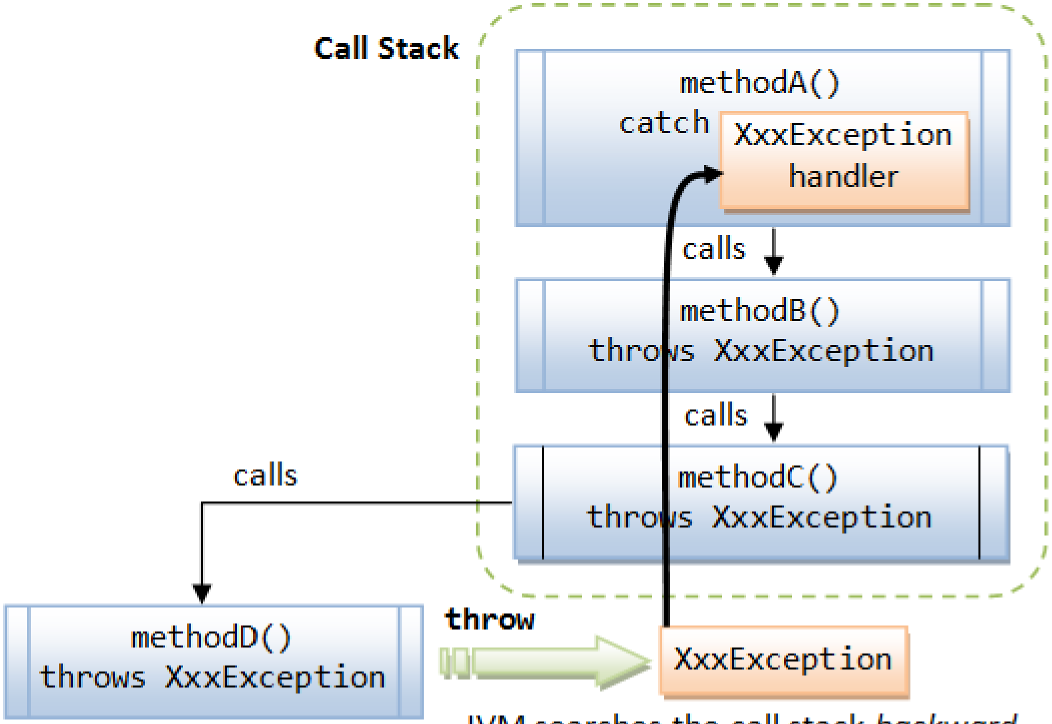
Exception & Call Stack

- This process is illustrated as follows. Suppose that **methodD()** encounters an abnormal condition and throws a **XxxException** to the JVM. The **JVM** searches **backward** through the call stack for a **matching exception handler**. It finds methodA() having a XxxException handler and passes the exception object to the handler.
- **Notice that methodC() and methodB() are required to declare "throws XxxException" in their method signatures in order to compile the program. (Relay of throwing exceptions)**

Exception Handling Mechanism

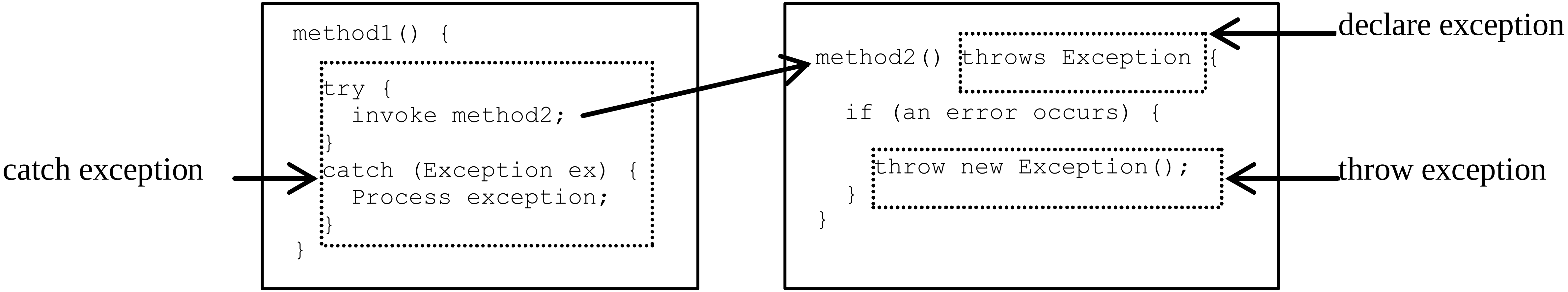


Method Call Stack
(Last-in-First-out Queue)



JVM searches the call stack *backward* for a *matching* exception handler

Declaring, Throwing, and Catching Exceptions



```
method1() {

  try {
    invoke method2;
  }
  catch (Exception ex) {
    Process exception;
  }
}
```

```
method2() throws Exception {

  if (an error occurs) {

    throw new Exception();
  }
}
```

catch exception

declare exception

throw exception

(The exception can be checked or un-checked.)

Declaring Exceptions

Every method must state the types of checked exceptions it might throw. This is known as *declaring exceptions*.

```
public void myMethod()
    throws IOException

public void myMethod()
    throws IOException, OtherException
```

Throwing Exceptions

When the program detects an error, the program can create an instance of an appropriate exception type and throw it. This is known as *throwing an exception*. Here is an example,

```
throw new TheException();

TheException ex = new TheException();
throw ex;
```

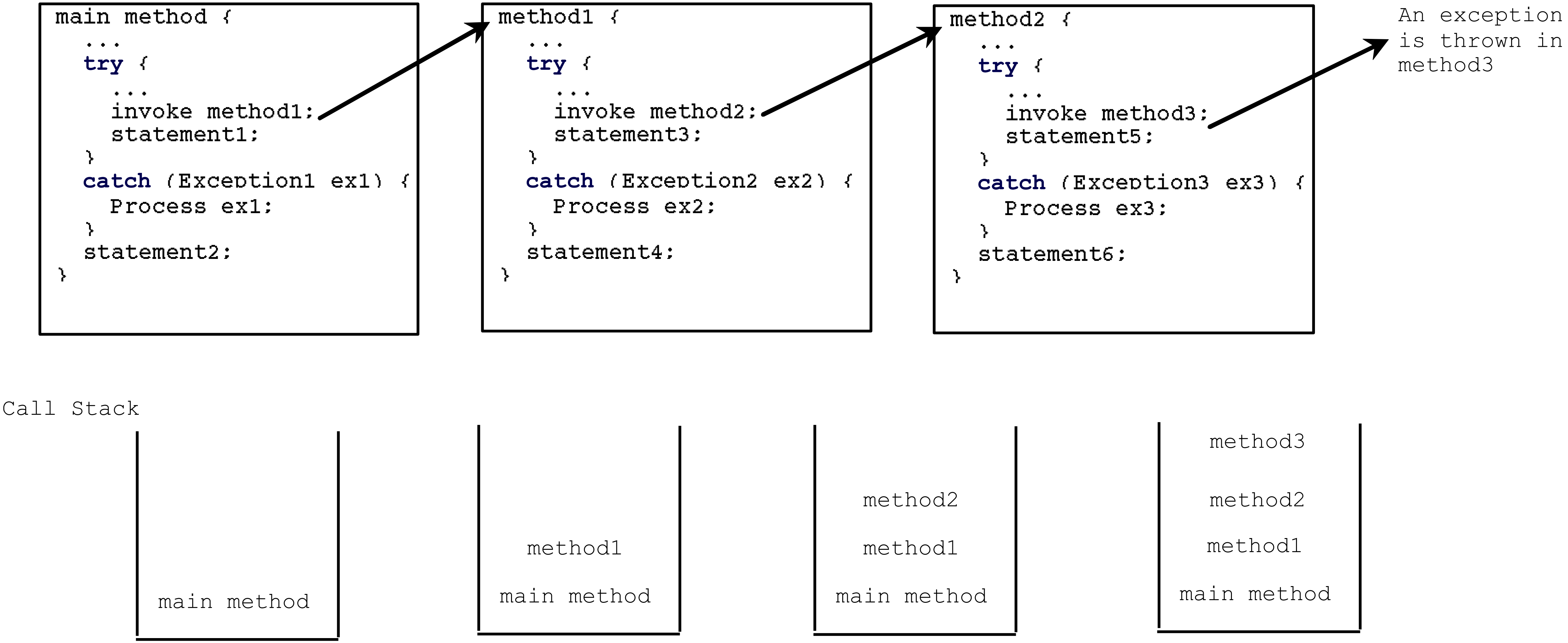
Throwing Exceptions Example

```java
   /** Set a new radius */
public void setRadius(double newRadius)
    throws IllegalArgumentException {
  if (newRadius >= 0)
    radius =  newRadius;
  else
    throw new IllegalArgumentException(
      "Radius cannot be negative");
}
```

Catching Exceptions

```
try {
  statements;  // Statements that may throw exceptions
}
catch (Exception1 exVar1) {
  handler for exception1;
}
catch (Exception2 exVar2) {
  handler for exception2;
}
...
catch (ExceptionN exVar3) {
  handler for exceptionN;
}
```

Catching Exceptions



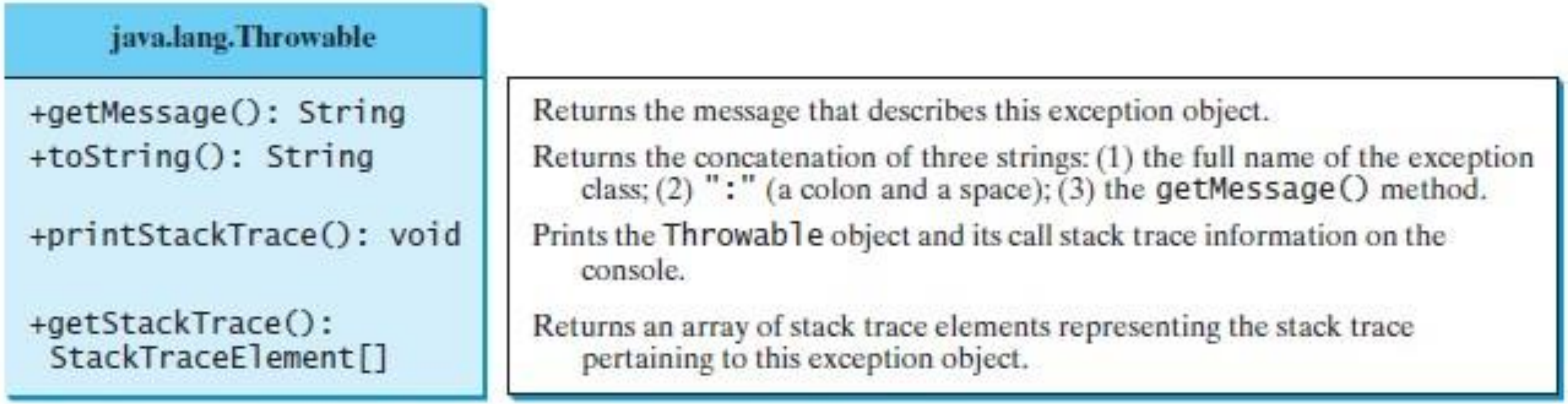
```
main method {
  ...
  try {
    ...
    invoke method1;
    statement1;
  }
  catch (Exception1 ex1) {
    Process ex1;
  }
  statement2;
}
```

```
method1 {
  ...
  try {
    ...
    invoke method2;
    statement3;
  }
  catch (Exception2 ex2) {
    Process ex2;
  }
  statement4;
}
```

```
method2 {
  ...
  try {
    ...
    invoke method3;
    statement5;
  }
  catch (Exception3 ex3) {
    Process ex3;
  }
  statement6;
}
```

An exception is thrown in method3

Throwable Class and Exception Class



java.lang.Throwable

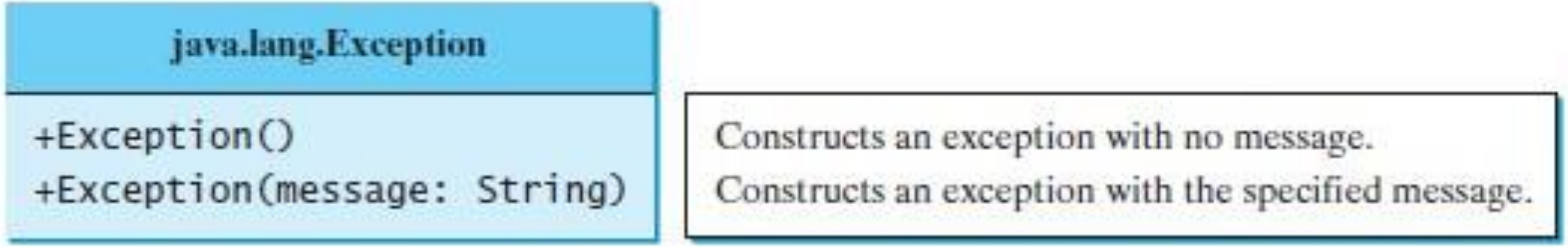
```
+getMessage(): String
+toString(): String

+printStackTrace(): void

+getStackTrace():
 StackTraceElement[]
```

Returns the message that describes this exception object.

Returns the concatenation of three strings: (1) the full name of the exception class; (2) ": " (a colon and a space); (3) the getMessage() method.

Prints the Throwable object and its call stack trace information on the console.

Returns an array of stack trace elements representing the stack trace pertaining to this exception object.



java.lang.Exception

```
+Exception()
+Exception(message: String)
```

Constructs an exception with no message.

Constructs an exception with the specified message.

Demonstration Program

TESTEXCEPTION.JAVA

Catch or Declare Checked Exceptions

- Java forces you to deal with checked exceptions. If a method declares a checked exception (i.e., an exception other than Error or RuntimeException), you must invoke it in a try-catch block or declare to throw the exception in the calling method. For example, suppose that method p1 invokes method p2 and p2 may throw a checked exception (e.g., IOException), you have to write the code as shown in (a) or (b).

```
void p1() {
  try {
    p2();
  }
  catch (IOException ex) {
    ...
  }
}
```

(a)

```
void p1() throws IOException {

  p2();

}
```

(b)

Example: Declaring, Throwing, and Catching Exceptions

- **Objective:** This example demonstrates declaring, throwing, and catching exceptions by modifying the <u>setRadius</u> method in the <u>Circle</u> class defined in Chapter 8. The new <u>setRadius</u> method throws an exception if radius is negative.

Demonstration Program

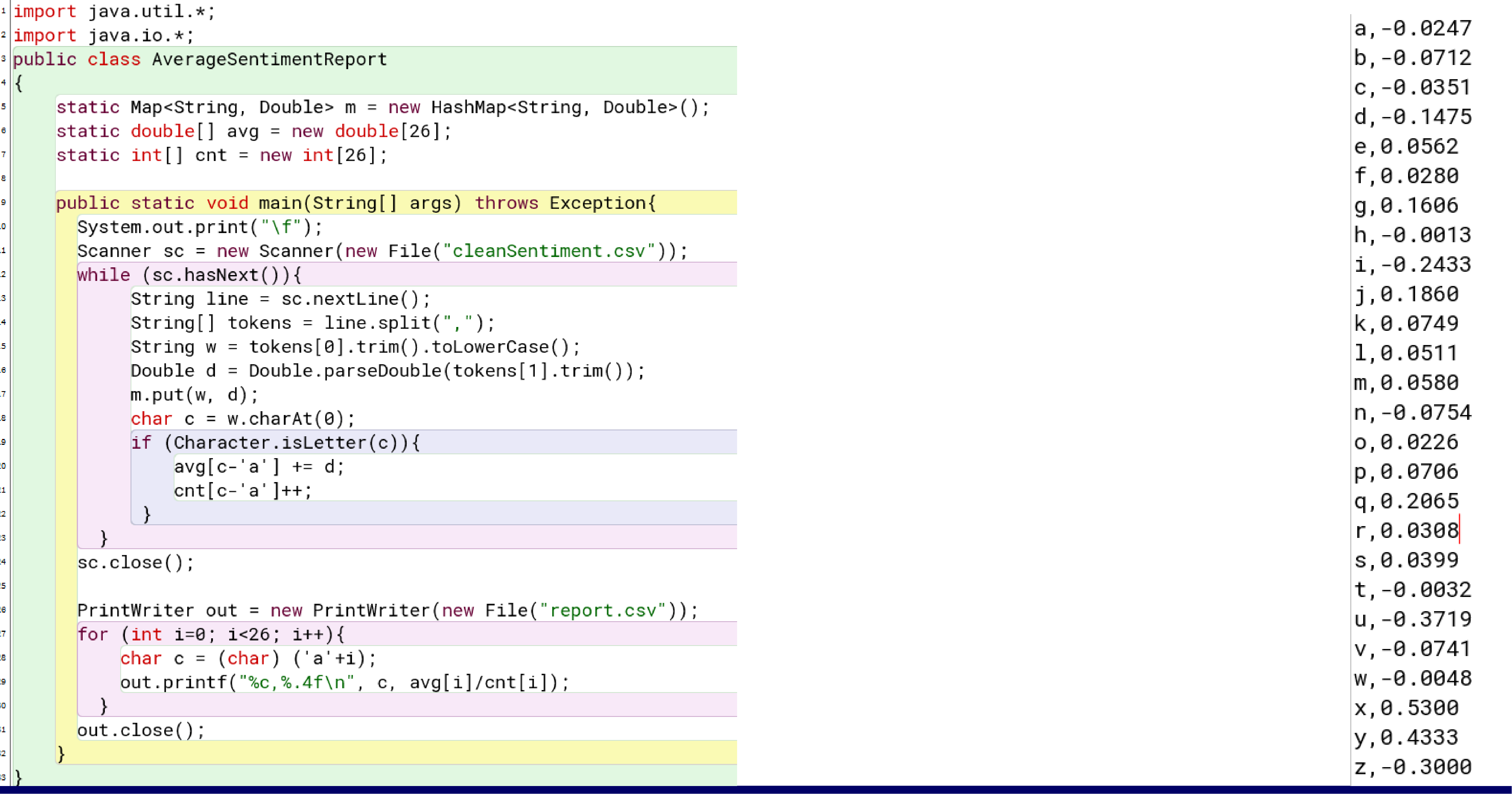
CIRCLEWITHEXCEPTION.JAVA
TESTCIRCLEWITHEXCEPTION.JAVA

File Output

LECTURE 12

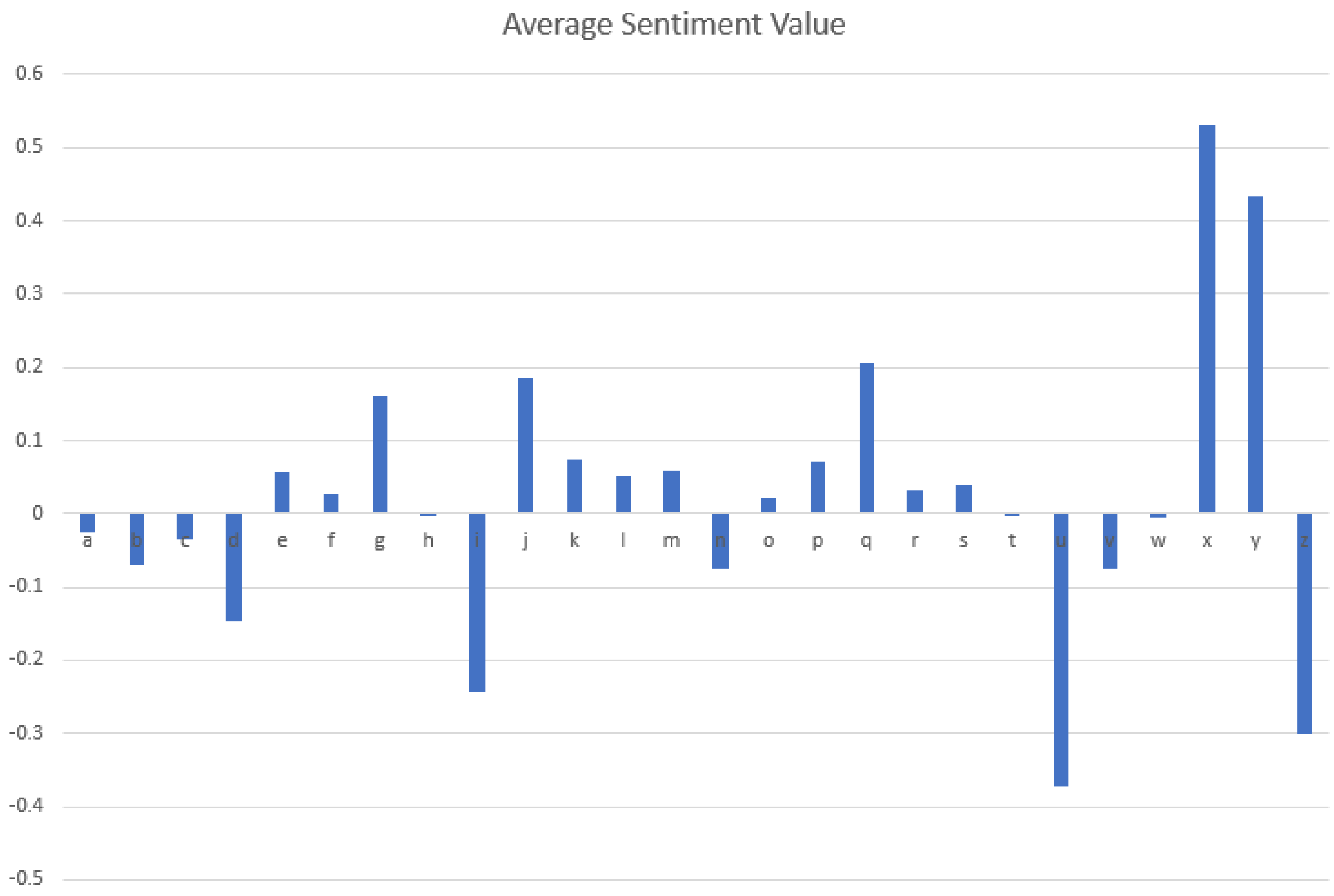
Calculate Average Sentiment Value for Each Starting Letter

- Analyze on the **cleanSentiment** data set. Calculate the average sentiment value for all words starting with the same letter.
- Use histogram.
- Output to a .csv file.



```java
import java.util.*;
import java.io.*;
public class AverageSentimentReport
{
    static Map<String, Double> m = new HashMap<String, Double>();
    static double[] avg = new double[26];
    static int[] cnt = new int[26];

    public static void main(String[] args) throws Exception{
       System.out.print("\f");
       Scanner sc = new Scanner(new File("cleanSentiment.csv"));
       while (sc.hasNext()){
            String line = sc.nextLine();
            String[] tokens = line.split(",");
            String w = tokens[0].trim().toLowerCase();
            Double d = Double.parseDouble(tokens[1].trim());
            m.put(w, d);
            char c = w.charAt(0);
            if (Character.isLetter(c)){
                avg[c-'a'] += d;
                cnt[c-'a']++;
             }
          }
       sc.close();

       PrintWriter out = new PrintWriter(new File("report.csv"));
       for (int i=0; i<26; i++){
           char c = (char) ('a'+i);
           out.printf("%c,%.4f\n", c, avg[i]/cnt[i]);
          }
       out.close();
    }
}
```

```
a,-0.0247
b,-0.0712
c,-0.0351
d,-0.1475
e,0.0562
f,0.0280
g,0.1606
h,-0.0013
i,-0.2433
j,0.1860
k,0.0749
l,0.0511
m,0.0580
n,-0.0754
o,0.0226
p,0.0706
q,0.2065
r,0.0308
s,0.0399
t,-0.0032
u,-0.3719
v,-0.0741
w,-0.0048
x,0.5300
y,0.4333
z,-0.3000
```


Average Sentiment Value
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z