# File Input
# (Token-Based Processing)

zyBook Chap 6.1, 6.4

# How do we locate a file?

- **Absolute Path** → specifies the file's location starting at the top-level folder
  - Linux/Macs → e.g., `"/Users/user1/CS1101/data.txt"`
  - Windows → e.g., `"C:/home/user1/CS1101/data.txt"`

**Always specify file paths using / in Java**

- **Relative Path** → does not specify any top-level folder
  - e.g., `"data.txt"`
  - When you construct a File object with a relative path, Java assumes it is **relative to the directory containing the program** (current directory)

# How do we read a file?

- **Step 1:** Specify the **file path** as a **String** object

```
String fileName = "data.txt";
```

- **Step 2:** Construct a **File** object to get the information about a file on the disk

```
import java.io.File;
File inputFile = new File(fileName);
```

- **Step 3:** Construct a **Scanner** object o read the file

```
import java.util.Scanner;
Scanner scnr = new Scanner(inputFile);
```

# Input from File

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

public class CourseNumber {
    public static void main (String[] args) {
        // Specify the file path (this example shows an absolute path)
        String fileName = "/Users/ginabai/Desktop/Spring23/LecExample/CourseNum.txt";

        // Construct a Scanner object to process the File object
        Scanner scnr = new Scanner( new File(fileName) );

        // Since the input comes from a file, we don't prompt the user for input
        int course = scnr.nextInt();

        System.out.println("The course number is " + course + ".");
        scnr.close();
    }
}
```

```
$ javac CourseNumber.java
CourseNumber.java:10: error: unreported exception FileNotFoundException;
must be caught or declared to be thrown
        Scanner scnr = new Scanner( new File(fileName) );
                                  ^
1 error
```

# Exception

- An **object** that **represents a program error** and prevents a program from continuing normal execution
  - Needs to be **import**ed

- Programs with invalid logic will cause exceptions.
  - dividing by 0
    - `ArithmeticException`
  - calling charAt() on a String and passing too large an index
    - `StringIndexOutOfBoundsException`
  - **trying to read a file that does not exist**
    - **`FileNotFoundException`**
  - …

# `FileNotFoundException`

It is a **checked exception** (checked at compile time)

- An error that must be handled by our program, otherwise it won't compile
- We must specify **what our program will do** to handle any **potential** file I/O failures.

**Some unchecked exceptions:**

- ArithmeticException
- StringIndexOutOfBoundsException
- NoSuchElementException
- InputMismatchException
- …

# Handling Exceptions

- Exceptions are **thrown** by the program
- We must either
  1) declare that our program will handle ("**catch**") the exception, OR
  2) state that we **choose not to handle the exception** and we accept that the program will crash if an exception occurs by **declaring the exception** in the **header of the method** that might generate it

```
$ javac CourseNumber.java
CourseNumber.java:10: error: unreported exception FileNotFoundException;
must be caught or declared to be thrown
        Scanner scnr = new Scanner( new File(fileName) );
                       ^
1 error
```

# `throws` Clause

A `throws` clause is a declaration that a method **may exit unexpectedly** due to a **particular type of exception**.

- It is an explicit acknowledgment that a statement in the method may throw the exception.
- A waiver of liability: "I hereby agree that this method might throw an exception, and I accept the consequences (crashing) if this happens."

```
public static <return type> <methodName>(<params>) throws <ExceptionType> {
        <statements>;
        ...
}
```

# Input from File

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class CourseNumber {
    public static void main (String[] args) throws FileNotFoundException {
        // Specify the file path (this example shows an absolute path)
        String fileName = "/Users/ginabai/Desktop/Spring23/LecExample/CourseNum.txt";

        // Construct a Scanner object to process the File object
        Scanner scnr = new Scanner( new File(fileName) );

        // Since the input comes from a file, we don't prompt the user for input
        int course = scnr.nextInt();

        System.out.println("The course number is " + course + ".");
        scnr.close();
    }
}
```

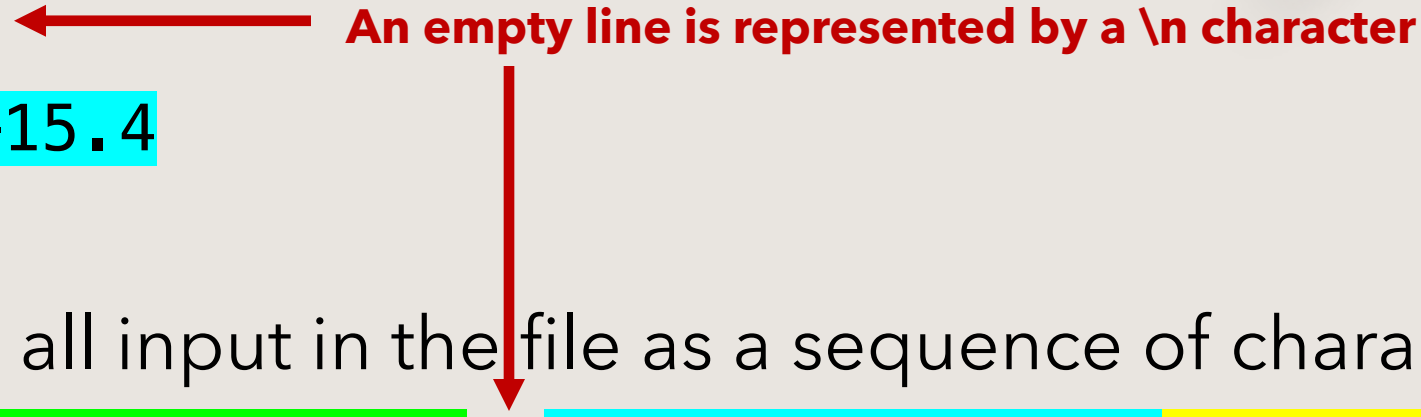
```
$ javac CourseNumber.java
$ java CourseNumber
The course number is 1101.
```

# Files and Input Cursor

- Consider a file CursorNum.txt that contains this text:

```
308.2
    14.9 7.4  2.8

3.9 4.7    -15.4
    2.8
```




- A Scanner sees all input in the file as a sequence of characters:

```
308.2\n\t14.9 7.4  2.8\n\n3.9 4.7\t-15.4\n\t2.8\n
```

# Recap – Token

A unit of user input, separated by whitespace.

- Each call to `nextInt()/nextDouble()/next()/nextLine()` advances the position of the scanner to the end of the current token, skipping over any whitespace

Example: How many tokens are there in the following file?

```
23 3.1
  "CS 1101"
```

| Token | Potential Input Type(s) |
|---|---|
| 23 | int, double, String |
| 3.1 | double, String |
| "CS | String |
| 1101" | String |

A Scanner always keeps track of its **current location** in a file using an **input cursor**. The cursor **starts at the beginning of the file**.

```
308.2\n\t14.9 7.4  2.8\n\n3.9 4.7\t-15.4\n\t2.8\n
```

```
scnr.nextDouble();
308.2\n\t14.9 7.4  2.8\n\n3.9 4.7\t-15.4\n\t2.8\n
```

```
scnr.nextDouble();
308.2\n\t14.9 7.4  2.8\n\n3.9 4.7\t-15.4\n\t2.8\n
```

# Recap – Scanner Exceptions

What happens when we reach the end of a file and we call `next()`?

- A **`NoSuchElementException`** is thrown signaling that there are no more tokens to process.
  - We can use the **`hasNext`** methods to check for the next tokens

What happens if you read a token as the wrong type?

- A **`InputMismatchException`** is thrown

# Scanner – Dealing with Files

- Scanners are designed for file processing in a **forward** manner.
  - **There is no support for reading the input backward.**
  - What if I need to read the information from the same file again?
    - Construct another Scanner object to process the same File object, which would position the input cursor at the beginning of the file.

# Scanner – Dealing with Files

- Scanners should be **closed** to avoid memory leaks.
  - **Every Scanner to an input file must be closed after you are done using it.**
  - Closing a Scanner to the console is NOT necessary, but it is a good programming practice.

- **Note**: If a Scanner object is closed and the program tries to use it, an `IllegalStateException: Scanner closed` would be thrown.

# Coding Practice

CourseDescription.txt

```
CS 1101 - Programming and Problem Solving

An intensive introduction to algorithm development and problem solving
on the computer. Structured problem definition, top down and modular
algorithm design. Running, debugging, and testing programs. Program
documentation. Not open to students who have earned credit for CS 1104
without permission. Total credit for this course and CS 1104 will not
exceed 3 credit hours. Credit hours reduced from second course taken
(or from test or transfer credit) as appropriate. FALL, SPRING. [3]
```

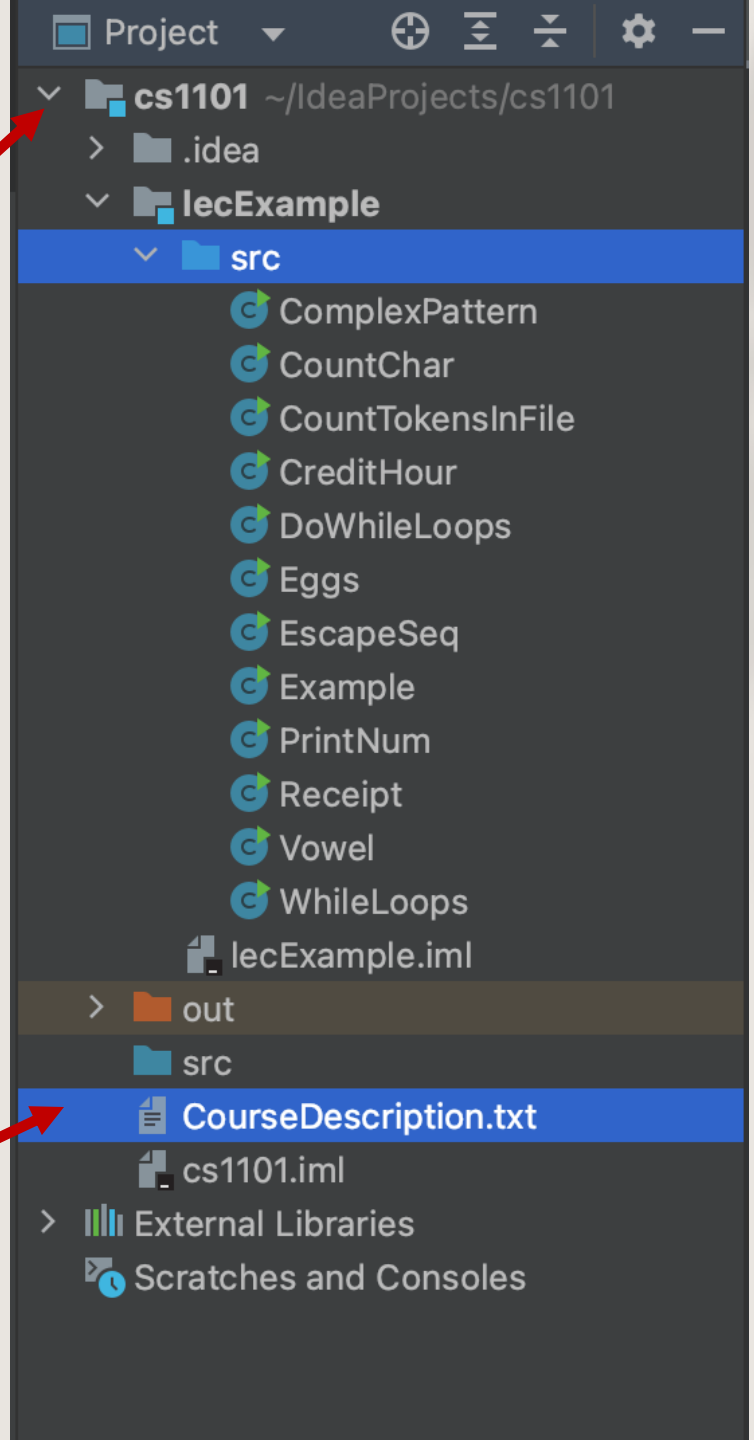
Write a program that reads in the file CourseDescription.txt, and

- counts the number of tokens in the file
- counts the number of integers in the file

```
$ javac CountTokensInFile.java
$ java CountTokensInFile
CourseDescription.txt has 80 token(s), including 4 integer(s).
```

# IDE Setup

- Place the input files in the **root directory**, e.g., cs1101



# Sample Solution

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class CountTokensInFile {
    public static void main (String[] args) throws FileNotFoundException {

        String fileName = "CourseDescription.txt"; // Relative path
        Scanner input = new Scanner(new File(fileName));

        int countToken = 0, countInt = 0;

        while (input.hasNext()) {
            if (input.hasNextInt()) {
                ++countInt;
            }
            input.next(); // Consume the token, so we can move to the next one
            ++countToken;
        }
        input.close(); // Close the scanner

        System.out.println(fileName + " has " + countToken + " token(s), including " +
                countInt + " integer(s).");
    }
}
```