LEC 01

CSE 122

File I/O – Token and line-based processing

Questions during Class?

Raise hand or send here

sli.do   **#cse122**

BEFORE WE START

Talk to your neighbors:

What's the latest show you have been obsessed with?

Instructors: Ido Avnon

TAs: Abby Williams
Chloë Mi Cartier
Connor Sun
Cynthia Pan
Katharine Zhang
Marcus Sanches
Rohini Arangam

Lecture Outline

- **Announcements/Reminders**

- Scanners for User Input and Files

  - Token-based & Line-based processing

- File I/O Examples

Announcements

- The IPL is open!
  - MGH 334
  - Schedule is on the course website; staffed by our awesome TAs!
  - Open 12:30 to 5:30PM most days, but check the schedule…

- Culminating Project 0 due Thursday, June 27th at 11:59pm
  - Check out the examples and resources!

- Make sure you can make it to quiz/exam days (Ed announcement)

- Check out PCMs before lecture!

- Go to your quiz sections!

Lecture Outline

- Announcements/Reminders

- **Scanner for User Input and Files**

  - **Token-based & Line-based Processing**

- File I/O Examples

(Review) Scanner for User input

Scanner is defined in the
`java.util` package

`import java.util.*;`

`Scanner console = new Scanner(System.in);`

Scanner Methods	Description
`nextInt()`	Reads the next token from the user as an `int` and returns it
`nextDouble()`	Reads the next token from the user as a `double` and returns it
`next()`	Reads the next token from the user as a `String` and returns it
`nextLine()`	Reads an *entire line* from the user as a `String` and returns it
`hasNextInt()`	Returns `true` if the next token can be read as an `int`, `false` otherwise
`hasNextDouble()`	Returns `true` if the next token can be read as a `double`, `false` otherwise
`hasNext()`	Returns `true` if there is another token of input to be read in, `false` otherwise
`hasNextLine()`	Returns `true` if there is another line of input to be read in, `false` otherwise

(PCM) **Token** vs. **Line-based Scanning**

The quick, brown fox
Jumped over the
Lazy dog.

Token are units of input (as defined by the Scanner) that are separated by *whitespace* (spaces, tabs, new lines)

(PCM) Token vs. Line-based Scanning

The quick,   brown fox
Jumped   over the
Lazy dog.

The

(PCM) <u>Token</u> vs. Line-based Scanning

The `quick,` brown fox
Jumped over the
Lazy dog.

quick,

(PCM) **Token** vs. Line-based Scanning

The quick, brown fox
Jumped   over the
Lazy dog.

brown

(PCM) **Token** vs. Line-based Scanning

The quick, brown fox

Jumped   over the

Lazy dog.

fox

(PCM) Token vs. <u>Line-based</u> Scanning

```
The quick,  brown fox
Jumped       over the
Lazy dog.
```

(PCM) Token vs. <u>Line-based</u> Scanning

The quick, brown fox
Jumped   over the
Lazy dog.

The quick, brown fox

Practice : Think

How many tokens are in the following line?

`"Hello world !" my-name is Ido`

A) Four   **B)** Five   **C)** Six   **D)** Seven

Practice : Pair

sli.do   **#cse122**

How many tokens are in the following line?

```
"Hello world !" my-name is Ido
```

A) Four   **B)** Five   **C)** Six   **D)** Seven

(PCM) Scanner for File I/O

Scanner is defined in the `java.util` package
```
import java.util.*;
```

File is defined in the `java.io` package
```
import java.io.*;
```

```java
File file = new File("Example.txt");
Scanner fileScan = new Scanner(file);
```

Scanner Methods	Description
`nextInt()`	Reads the next token from the user as an `int` and returns it
`nextDouble()`	Reads the next token from the user as a `double` and returns it
`next()`	Reads the next token from the user as a `String` and returns it
`nextLine()`	Reads an *entire line* from the user as a `String` and returns it
`hasNextInt()`	Returns `true` if the next token can be read as an `int`, `false` otherwise
`hasNextDouble()`	Returns `true` if the next token can be read as a `double`, `false` otherwise
`hasNext()`	Returns `true` if there is another token of input to be read in, `false` otherwise
`hasNextLine()`	Returns `true` if there is another line of input to be read in, `false` otherwise

(PCM) Typical Line-Processing Pattern

```java
while (scan.hasNextLine()) {
    String nextLine = scan.nextLine();
    // do something with nextLine
}
```

(PCM) Typical Token-Processing Pattern

```
while (scan.hasNext__()) {
    __ nextToken = scan.next__();
    // do something with nextToken
}
```

(PCM) Checked Exceptions

If you try to compile a program working with file scanners, you may encounter this error message:

```
error: unreported exception FileNotFoundException; must be
caught or declared to be thrown
```

To resolve this, you need to be `throws FileNotFoundException` at the end of the header of any method containing file scanner creation code, or any method that calls that method!

This is like signing a waiver and telling Java – "Hey, I hereby promise to not get mad at you when you bug out and crash my program if I give you a file that doesn't actually exist."

Practice : Think

What is the output of this Java program?

```java
import java.util.*;
import java.io.*;
public class Demo {
    public static void main(String[] args) throws
                            FileNotFoundException {
        File f = new File("Example.txt");
        Scanner input = new Scanner(f);
        while (console.hasNextLine()) {
            System.out.print(console.nextLine() + ", ");
        }
    }
}
```

Example.txt:
```
One Two
Three
```

A) One, Two, Three,

B) One,   **C)** One Two,
Two,   Three,
Three,

D) One Two, Three,

E) Error / Exception

What is the output of this Java program?

```java
import java.util.*;
import java.io.*;
public class Demo {
    public static void main(String[] args) throws
                            FileNotFoundException {
        File f = new File("Example.txt");
        Scanner input = new Scanner(f);
        while (console.hasNextLine()) {
            System.out.print(console.nextLine() + ", ");
        }
    }
}
```

Example.txt:

```
One Two
Three
```

A) One, Two, Three,

B) One,   C) One Two,
Two,   Three,
Three,

D) One Two, Three,

E) Error / Exception

Lecture Outline

- Announcements/Reminders

- Review Java

- Scanner for User Input and Files

    - Token-based & Line-based Processing

- **File I/O Examples** ◀

(Friday's PCM) Typical Hybrid Pattern

```java
while (fileScan.hasNextLine()) {
    String line = fileScan.nextLine();
    Scanner lineScan = new Scanner(line);
    while (lineScan.hasNext__()) {
        __ nextToken = lineScan.next__();
        // do something with nextToken
    }
}
```