

LEC 02

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 your favorite YouTube or Twitch channel to watch?


Music: [122 24wi Lecture Tunes](#) 

Instructors Miya Natsuhara and Joe Spaniac

TAs

Ailsa	Chaafen	Helena	Megana	Sahej
Alexander	Chloe	Jessie	Mia	Shivani
Ambika	Claire	Katharine	Minh	Smriti
Andy	Colin	Kavya	Nicolas	Steven
Arkita	Colton	Ken	Poojitha	Vinay
Atharva	Connor	Kyle	Rohini	Zane
Autumn	Elizabeth	Logan	Ronald	
Ayush	Hannah	Marcus	Rucha	


Lecture Outline

- **Announcements/Reminders** 
- Review Java
- 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 9:30PM most days, but check the schedule...
- Creative Project 0 due Thursday, January 11th at 11:59pm
 - Make sure to go to the “Final Submission” slide and submit!
 - Submit as many times as you’d like—we will only grade the latest submission made *before the deadline*
- Just joined CSE 122? That’s okay; look at Ed & course website and catch up!
 - Freaking out that C0 is due this Thursday? It’s ok! Resubmission cycles allow you to submit it later

Lecture Outline

- Announcements/Reminders
- **Review Java** 
- Scanners for user input and Files
 - Token-based & Line-based Processing
- File I/O Examples


Reminders: Review Java Syntax

[Java Tutorial](#) reviews all the relevant programming features you should familiar with (even if you don't know them in Java).

- Printing and comments
- Variables, types, expressions
- Conditionals (if/else if/ else)
- Loops (for and while)
- Strings
- Methods
- Arrays & 2D arrays

Recording of the [Java Review Session](#) from Monday (January 8th) are posted on the course calendar!

Lecture Outline

- Announcements/Reminders
- Review Java
- **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 <code>int</code> and returns it
nextDouble()	Reads the next token from the user as a <code>double</code> and returns it
next()	Reads the next token from the user as a <code>String</code> and returns it
nextLine()	Reads an <i>entire line</i> from the user as a <code>String</code> and returns it
hasNextInt()	Returns <code>true</code> if the next token can be read as an <code>int</code> , <code>false</code> otherwise
hasNextDouble()	Returns <code>true</code> if the next token can be read as a <code>double</code> , <code>false</code> otherwise
hasNext()	Returns <code>true</code> if there is another token of input to be read in, <code>false</code> otherwise
hasNextLine()	Returns <code>true</code> if there is another line of input to be read in, <code>false</code> otherwise

(PCM) Token vs. Line-based User input

The quick, brown fox
Jumped over the
Lazy dog.

(PCM) Token vs. Line-based User input

The quick, brown fox
Jumped over the
Lazy dog.

The

(PCM) Token vs. Line-based User input

The quick, brown fox
Jumped over the
Lazy dog.

quick,

(PCM) Token vs. Line-based User input

The quick, brown fox
Jumped over the
Lazy dog.

brown

(PCM) Token vs. Line-based User input

The quick, brown fox
Jumped over the
Lazy dog.

fox

(PCM) Token vs. Line-based User input

The quick, brown fox
Jumped over the
Lazy dog.

(PCM) Token vs. Line-based User input

The quick, brown fox

I Jumped over the

Lazy dog.

The quick, brown fox



Practice : Think



sli.do

#cse122

How many tokens are in the following line?

“Hello world !” my-name is Joe

- 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 Joe

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

(Review) Typical Token-Processing Pattern

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

(Review) Typical Line-Processing Pattern

```
Scanner scan = new Scanner(System.in);
while (scan.hasNextLine()) {
    String nextLine = scan.nextLine();
    // do something with nextLine
}
```

(PCM) Scanner & File for File input

File is defined in the `java.io` package:

```
import java.io.*;
```

```
File newFile = new File("newFile.txt");
```

```
Scanner fileScan = new Scanner(newFile);  
while (fileScan.hasNextLine()) {  
    String line = fileScan.nextLine();  
    // do something with line  
}
```

(PCM) Typical Line-Processing Patterns

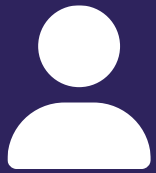
Console Input:

```
Scanner console = new Scanner(System.in);
while (console.hasNextLine()) {
    String line = console.nextLine();
    // do something with line
}
```

File Input:

```
File newFile = new File("newFile.txt");
Scanner fileScan = new Scanner(newFile);
while (fileScan.hasNextLine()) {
    String line = fileScan.nextLine();
    // do something with line
}
```

Notice any similarities between the two?



Practice : Think

[sli.do](#)[#cse122](#)

What is the output of this Java program?

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

A) One, Two, Three,

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

D) One Two, Three,

E) Error / Exception

Example.txt:

One Two
Three



Practice : Pair



sli.do

#cse122

What is the output of this Java program?

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

- A) One, Two, Three,
- B) One, C) One Two,
Two, Three,
Three,
- D) One Two, Three,
- E) Error / Exception

Example.txt:

One Two
Three

(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."

Lecture Outline

- Announcements/Reminders
- Review Java
- Scanner for User Input and Files
 - Token-based & Line-based Processing
- **File I/O Examples** 