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:

Slido Poll: Apple pie or pumpkin pie?

Music: 122 25au Lecture Tunes

Instructor: Elba Garza

TAs:

Sreshta	Merav	Shivani	Yang
William	Nicole	Naomi	Cady
Arjun	Vrinda	Hanna	Diya
Dani	Shreya	David	Katharine
Rohan	Wesley	Sushma	
Andrew	Isis	Rio	
Saachi	Colin	Nicolae	
Ava	Medha	Ivory	

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, October 2nd at 11:59pm PT
  - Make sure to complete 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 tomorrow? It's ok! Resubmission cycles allow you to submit it later.
- Go to your designated quiz sections!
- Quiz dates:

Quiz 0: October 14th **Quiz 1**: November 4th **Quiz 2**: November 20th

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

Recordings of both sessions are now available on the course website in 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 `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.
```

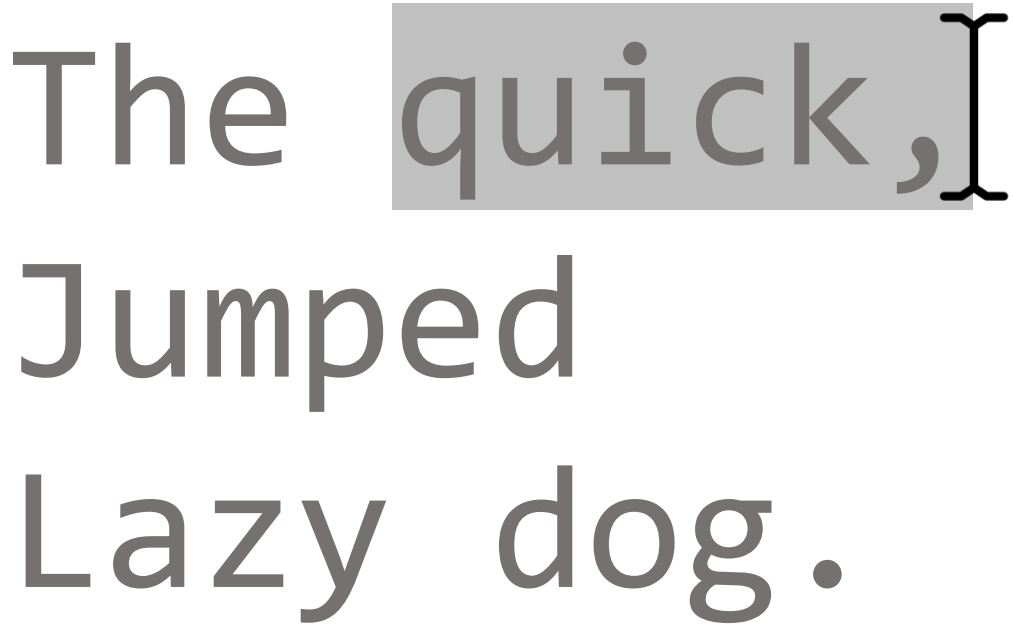
Token are units of input (as defined by the Scanner) that are separated by *whitespace* (spaces, tabs, new lines)

(PCM) Token vs. Line-based Scanning

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

The

(PCM) Token vs. Line-based Scanning



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. Line-based Scanning

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

etc.

(PCM) Token vs. Line-based Scanning

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

(PCM) Token vs. Line-based Scanning

```
The quick,  brown fox
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 Elba
```

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 Elba
```

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.*;
```

```
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) 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 and crash my program if I give you a file that doesn't actually exist."

(PCM) Typical Line-Processing Pattern

```
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
}
```

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) throws
                                    FileNotFoundException {
      File f = new File("Example.txt");
      Scanner fileScan = new Scanner(f);
      while (fileScan.hasNextLine()) {
         System.out.print(fileScan.nextLine() + ", ");
      }
   }
}
```

Example.txt:

```
One Two
Three
```

A) One, Two, Three,

B) One, Two, Three,

C) One Two, Three,

D) One Two, Three,

E) Error / Exception

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) throws
                                    FileNotFoundException {
      File f = new File("Example.txt");
      Scanner fileScan = new Scanner(f);
      while (fileScan.hasNextLine()) {
         System.out.print(fileScan.nextLine() + ", ");
      }
   }
}
```

Example.txt:

```
One Two
Three
```

A) One, Two, Three,

B) One, Two, Three,

C) One Two, 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

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