

CIS 110: Introduction to Computer Programming

Lecture 15
Our Scanner eats files
(§ 6.1-6.2)

10/31/2011

CIS 110 (11fa) - University of Pennsylvania

1

Outline

- Programming assertion recap
- The Scanner object and files
- Token-based file processing

10/31/2011

CIS 110 (11fa) - University of Pennsylvania

2

Exam announcements

- Attempting to reschedule midterm #2 to 11/21 (Monday of Thanksgiving break)
 - Let me know asap if you will be out of town.
- Final time has been *confirmed* for 12/19, 6-8 PM
 - Let me know asap if you need to reschedule.

10/31/2011

CIS 110 (11fa) - University of Pennsylvania

3

Programming assertions revisited

10/31/2011

CIS 110 (11fa) - University of Pennsylvania

4

An extended example: mystery

```
public static int mystery(int n) {
    int x = 0;
    // Point A
    if (n < 0) { return -1; }
    while (n != 0) {
        // Point B
        int d = n % 10;
        if (d % 2 == 1) {
            x += d;
        }
        // Point C
        n /= 10;
    }
    // Point D
    return x;
}
```

For each point, are the following always/sometimes/never true?
 1) $n < 0$
 2) $x \geq 0$
 3) $d < 10$
 4) $x < n$
 (See AssertionProblem.java.)

University of Pennsylvania

5

The Scanner object and files

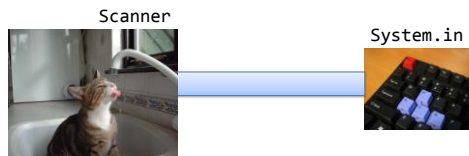
10/31/2011

CIS 110 (11fa) - University of Pennsylvania

6

Scanners revisited

- A Scanner is a *faucet* over some pipe of data.



10/31/2011

CIS 110 (11fa) - University of Pennsylvania

7

Empty pipes

- If the pipe is empty, the scanner first gets a line of input from the user, e.g., one call to `next()`.



10/31/2011

CIS 110 (11fa) - University of Pennsylvania

8

Consuming input from the pipe

- The call to `next()` then consumes the first *token* of input.



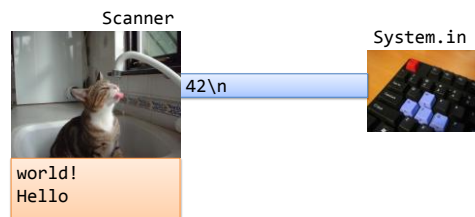
10/

110 (11fa) - University of Pennsylvania

9

Consuming tokens

- When we consume a token, there's no way to "go back", only forward!



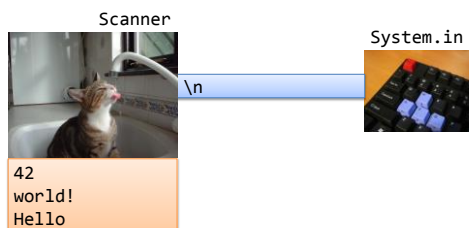
10/

110 (11fa) - University of Pennsylvania

10

Consuming tokens as different types

- We can consume a token *and* translate it to a particular type, e.g., `nextInt()`.



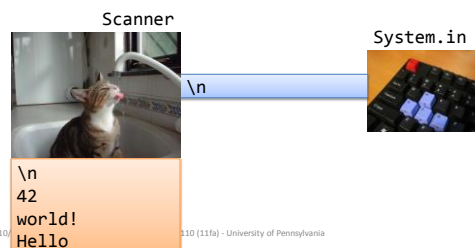
10/

110 (11fa) - University of Pennsylvania

11

Consuming the rest of a line

- We can consume the rest of a line with `nextLine()`.



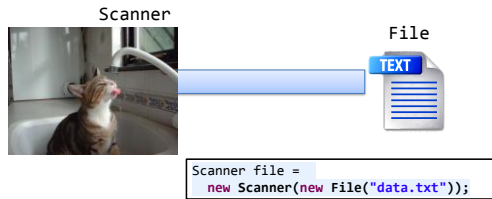
10/

110 (11fa) - University of Pennsylvania

12

Plugging in different data sources

- A Scanner can accept many kinds of data sources such as Files instead!



10/31/2011

CIS 110 (11fa) - University of Pennsylvania

13

The File object

- A File object represents a *file or directory on disk*.
 - Exists in the `java.io` package.

```
File file = new File("data.txt");
System.out.println("canRead? " + file.canRead());
System.out.println("exists? " + file.exists());
// Renames the file to the given file's name.
file.renameTo(new File("foo.txt"));
// Deletes the file from disk if it exists.
file.delete();
```



10/31/2011

CIS 110 (11fa) - University of Pennsylvania

14

An exceptional problem

```
public static void main(String[] args) {
    Scanner file = new Scanner(new File("data.txt"));
}
```

- The following code fails to compile. Why?
 - "unreported exception `java.io.FileNotFoundException`; must be caught or declared to be thrown"
- Example of a *checked exception* in Java.

10/31/2011

CIS 110 (11fa) - University of Pennsylvania

15

Checked and unchecked exceptions

- Java distinguishes between two sorts of exceptions.
 - Unchecked exceptions* represent program bugs
 - Checked exceptions* represent badness outside of the program's control.

Unchecked exceptions

```
IndexOutOfBoundsException
IllegalArgumentException
StackOverflowError
```

Checked exceptions

```
FileNotFoundException
```

10/31/2011

CIS 110 (11fa) - University of Pennsylvania

16

Dealing with checked exceptions

- Two solutions:
 - Annotate the enclosing method with a *throws clause*.
 - Use a *try-catch block*.

```
public static void main(String[] args)
    throws FileNotFoundException {
    Scanner file =
    new Scanner(new File("data.txt"));
}
```

```
try {
    Scanner file = new Scanner(
        new File("data.txt"));
} catch (FileNotFoundException ex) {
    ex.printStackTrace();
}
```

Which do we use!?

10/31/2011

CIS 110 (11fa) - University of Pennsylvania

17

Checked exceptions: a holy war

- Big debate if checked exceptions are "worth it".
- General advice: use try-catch when you can do something meaningful with the exception.
 - Give a good error message, re-throw, etc.
- For this class: we'll use *throws clauses*.

```
public static void main(String[] args)
    throws FileNotFoundException {
    Scanner file =
    new Scanner(new File("data.txt"));
}
```

10/31/2011

CIS 110 (11fa) - University of Pennsylvania

18

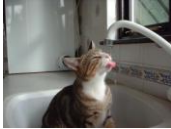
Token-based processing

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 19

Abstraction at its finest


- The methods of the Scanner we've learned so far apply when we use a File instead!

Scanner



→

File



```
Scanner file =
new Scanner(new File("data.txt"));
```

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 20

File processing example: FileSum


```
import java.util.*;
// Necessary since FileNotFoundException is also in java.io.
import java.io.*;

public class FileSum {
    public static void main(String[] args)
        throws FileNotFoundException {
        Scanner file = new Scanner(new File("data.txt"));
        double sum = 0.0;
        while(file.hasNextDouble()) {
            double d = file.nextDouble();
            System.out.println("Adding up " + d + "...");
            sum += d;
        }
        System.out.println("Total = " + sum);
    }
}
```

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 21

A file is just a long-ass string

data.txt



→

3.4 7.1 4.3
5.9 1.1 2.5
3.6 -1.2

↪


3.4 7.1 4.3\n 5.9 1.1 2.5\n\n3.6 -1.2\n

We can think of a file as a sequence of characters by replacing newlines with '\n'

```
while(file.hasNextDouble()) {
    double d = file.nextDouble();
}
```

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 22

Input cursor




Input cursor

3.4 7.1 4.3\n 5.9 1.1 2.5\n\n3.6 -1.2\n

The input cursor is our current position in the file, initially at the beginning.


data.txt



```
while(file.hasNextDouble()) {
    double d = file.nextDouble();
}
```

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 23

Input cursor and input consumption




Input cursor

3.4 7.1 4.3\n 5.9 1.1 2.5\n\n3.6 -1.2\n

On each call to nextDouble, we consume the next double and advance the input just past the token.

data.txt



```
while(file.hasNextDouble()) {
    double d = file.nextDouble();
}
```

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 24

Jumping that whitespace

3.4 7.1 4.3\n 5.9 1.1 2.5\n\n3.6 -1.2\n

↑
Input cursor

Calls to nextX()
skip whitespace to
the next token.

data.txt

```
while(file.hasNextDouble()) {
    double d = file.nextDouble();
}
```

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 25

Newlines are whitespace

3.4 7.1 4.3\n 5.9 1.1 2.5\n\n3.6 -1.2\n

↑
Input cursor

↓

3.4 7.1 4.3\n 5.9 1.1 2.5\n\n3.6 -1.2\n

↑
Input cursor

data.txt

```
while(file.hasNextDouble()) {
    double d = file.nextDouble();
}
```

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 26

hasNextX looks ahead

3.4 7.1 4.3\n 5.9 1.1 2.5\n\n3.6 -1.2\n

↑
Input cursor

hasNextDouble()
now returns false!

data.txt

```
while(file.hasNextDouble()) {
    double d = file.nextDouble();
}
```

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 27

Mixing up types

- We can mix different nextX functions as necessary.

Jerry 21 13.7 true

file.next();

file.nextInt();

file.nextDouble();

file.nextBooelean();

But we get
NoSuchElementException
if we're wrong!

10/31/2011 CIS 110 (11fa) - University of Pennsylvania 28