Lecture 8 – File & Network I/O

08-671
Java Programming for App Developers

September 24, 2015

Jeffrey L. Eppinger & Terry Lee

08-671 Lecture Topics

(subject to change – but only a little bit)

- #1 Intro
- #2 Primitive Types
- #3 Java Classes
- #4 Reference Types
- #5 Loops & Arrays
- #6 Methods & Classes
- #7 Lists & Maps
- #8 File & Network I/O
- #9 Swing Interfaces
- #10 Swing Actions
- #11 Threads
- #12 Exceptions
- #13 Functional Programming
- #14 In-class Written Exam

* Final Exam – this will be a 3-hour programming problem

Good News

- HW5 doesn’t have data load from files

Outline

- ✓ HW#5
- ⇢ Recap Lists & Maps
- Interfaces & Abstract Classes
- File I/O
- Network I/O
- Recitation

List Recap

- You know about arrays
- Lists let you store a collection of items

  …with a specific ordering

  – Many internal implementations are possible
    - We looked at ArrayList and LinkedList

  – Depending on the implementation, certain operations are faster than others

Performance Comparisons

	Append After Last	Insert Before First	Lookup by Position	Lookup by Value	Remove Last	Remove First
Array ArrayList	O(1)*	O(*n*)	O(1)	O(*n*)	O(1)	O(*n*)
LinkedList	O(1)	O(1)	O(*n*)	O(*n*)	O(1)	O(1)

* On average this operation will be constant O(1) time.

Maps Recap

- Maps let you find an object by a unique key
  - We declare it as a Map<K,V>
  - The usual implementation uses hash codes
    - We looked at HashMap
  - Get, put, and remove are all fast operations

  … but there is no ordering of the objects in the map
- Sets can be easily implemented using a map
  - We discussed HashSet

Performance Comparisons

	Append After Last	Insert Before First	Lookup by Position	Lookup by Value	Remove Last	Remove First
ArrayList	O(1)*	O(n)	O(1)	O(n)	O(1)	O(n)
LinkedList	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
HashSet HashMap	Add: O(1)		N/A	O(1)	Remove: O(1)	

* On average this operation will be constant O(1) time.

Outline

✓ HW#5

✓ Recap Lists & Maps

⇢ Interfaces & Abstract Classes

File I/O

Network I/O

Recitation

Java Interfaces

- A Java Interface allows you to specify methods that must be implemented by a class
  - It's just a list of methods, but no implementations
  - We looked at several interfaces
    - List<E>
    - Map<K,V>
    - Set<E>
    - Comparable<E>

Abstract Classes

- A class in the middle of class hierarchy
- Incomplete
  - Has methods & variables common to its subclasses
  - Some methods can be abstract (specification only)
- Must be subclassed (to be instantiated)
  - You can't say "new" on an abstract class

Abstract Class vs. Interfaces

- An abstract class can have some implemented methods
- An abstract class must be subclassed to be used (i.e., it must be "extended")
- An interface cannot have any implemented methods (it's totally abstract)
- An interface must be "implemented"

Abstract Classes for I/O

- In the `java.io` package
- The `InputStream` & `OutputStream` classes
  - Abstract Classes
  - Let you read & write bytes
- The `Reader` & `Writer` classes
  - Abstract Classes
  - Let you read & write characters
- For this lecture, we’ll just be looking at Text Files that contain only characters

Outline

- ✓ HW#5
- ✓ Recap Lists & Maps
- ✓ Interfaces & Abstract Classes
- ⇢ File I/O
- Network I/O
- Recitation

Byte I/O

- An `InputStream` … lets you read bytes
  - An abstract class
  - Interesting subclasses:
    - `FileInputStream`
    - `ByteArrayInputStream`
    - Many others…
- An `OutputStream` … lets you write bytes
  - Similar subclasses

Character I/O

- `Reader` & `Writer` classes added in Java 1.1 to handle characters
  - More efficient
  - Handle Unicode translations
- We will primarily use Readers and Writers

Examples using Readers

- Reads/prints out any file as characters:
  `FilePrint.java`
- Typical way to read a file:
  `ReadLineTest.java`
- Programs to demo speed of BufferedReader:
  `CountTest.java`
  `CountTestBuffered.java`
- If time, a program to find strings in any file:
  `HiddenStrings.java`

What Exceptions Must You Catch?

- Subclasses of **`RuntimeException`** and **`Error`** need not be caught or declared as thrown from a method
  - Examples:
    - **`NumberFormatException`**
    - **`ArrayIndexOutOfBounds`**
    - **`StackOverflowError`** (later)
- Other exceptions must be caught or your method declaration must state they you might throw them
  - Examples:
    - **`FileNotFoundException`**
    - **`IOException`**
    - **`InterruptedException`** (later)

Example Data Files

- A list of 10 Ticker/Share pairs representing the 100 shares in companies I wish I has bought when they went public:

    `100.csv`

- A snapshot of the data provided by Yahoo in the StockQuote class:

    `quotes.csv`

Let’s Subclass to Handle Parsing

- We can subclass BufferedReader to handle the parsing of the input stream
- Examples in this course are CSV data
  - (Comma separated values)

CSV Parsing Example

- Class to read CSV file and return the values separated in an array of Strings

  `CSVReader.java`

- And a Test Program to test CSVReader

  `CSVReaderTest.java`

Outline

- ✓ HW#5
- ✓ Recap Lists & Maps
- ✓ Interfaces & Abstract Classes
- ✓ File I/O
- ⇢ Network I/O
- Recitation

IP Addresses

- A unique number of each network address on the Internet
- IPv4
  - The 2nd generation of IP addresses
  - A four byte IP address, ~ a billion addresses, not all available
  - We're running out of them, well some people are
- IPv6
  - The 3rd generation of IP addresses
  - A sixteen byte IP address, literally zillions of addresses

Ports

- Your computer has a small number of IP addresses (typically: wired, wireless, localhost)
- Each application on your computer can use one or more port numbers to cause network traffic to be routed to it
- Example: netstat

Where to Read More on IP

- Check out all the entries on Wikipedia
- They are extensive and seem pretty accurate

How to read from the network

- Use the `java.net` package
  - Specifically, use `Socket` and `ServerSocket` classes
  - Servers bind and accept and then get the `InputStream` and `OutputStream`
  - Clients bind, connect and then get the `InputStream` and `OutputStream`
  - For character data, use your Streams to make Readers/ Writers
  - For Objects, use `ObjectInputStream` and `ObjectOutputStream`
- Examples: Server.java and Client.java

HTTP Protocol

- This is just a socket protocol
  - Runs on port 80 by default
  - Send HTTP commands (GET, POST, etc)
    - Send parameter (or post data)
  - Receive HTTP reply
    - Header
    - Data

How to read from the web in Java?

- Use `java.net.URL`
  - Get an `InputStream`
  - Get an `InputStreamerReader`
  - Get a `BufferedReader`
  - Read it like a file
- Don’t forget to close everything when done
- Examples
  - URLToStringTest.java
  - WebCrawler.java

Outline

- ✓ HW#5
- ✓ Recap Lists & Maps
- ✓ Interfaces & Abstract Classes
- ✓ File I/O
- ✓ Network I/O
- ⇢ Recitation

Recitation Tomorrow

- Null vs Empty String
- Speed of Data Structures (demo)
- Sorting using Comparators (demo)
- Use of hashing in switch
- NullPointerException
- AutoBoxing Examples (good and bad)
- HW#5 Strategies
- Immutable Objects
- Quiz#4