

# Introduction to Computation and Problem Solving

**Class 30:  
Lab: Streams  
November 29, 2005**

**Prof. Steven R. Lerman  
and  
Dr. V. Judson Harward**

## The Logger Application



## Goals

Using the framework of the Logger application, we are going to explore three ways to read and write data using Java streams:

1. as text
2. as binary data
3. as serialized objects

3

## Logger Architecture

`Logger.java`:

- provides `main()` and GUI
- generates log data ("Log") as `Date` objects
- summarizes and performs minimal validity check ("`Summarize`");
- selects `Files` for reading and writing using a `JFileChooser`
- calls methods in `LoggerIO` to perform reading and writing of data

You are going to write those methods.

4

## `java.util.LinkedList`

- `java.util.LinkedList` possesses the same methods as our `SLinkedList` and many others.
- `Logger` uses a `LinkedList` to hold the log data, and it passes the list in calls to the `LoggerIO` methods you will have to write. For instance:

```
void saveToText( List list, File f )
```

5

## Logger Exercise

- Download the two files `Logger.java` and `LoggerIO.java` in the .zip file `Lecture30.zip` from the class web site.
- Save them both to a new directory.
- Create a new project based on the directory into which you just saved the two java files. Compile the project and test it. Try to save to a file in text mode. Did you create a file? Was there anything in it? (Look at it with the Windows Notepad application).

6

## `saveToText ( )`

- We have given you almost the whole `saveToText ( )` method; get a sense of the pattern:
  - The try/catch blocks
  - How we construct the stream, a `FileWriter` in this case
  - How we iterate down the list
  - How we close the stream when we are done (closing the stream saves the file)
- Find the method in `FileWriter` to write the `dateString` and use it to output `nString` on writer.
- Test it. Look at what you wrote using Notepad.

7

## `loadFromText ( ) , 1`

- Now proceed to the `loadFromText ( )` method. Once again observe the pattern:
  - Construct a `BufferedReader` from a `FileReader`; we need a `BufferedReader` to read text a whole line at a time;
  - Look at the `while` condition; we read the next line and check for EOF in one statement. Many input stream read methods return a special value when they reach the end of file.
  - Understand why we use nested try/catch blocks and multiple catch clauses.
- Write a line of code to convert the line of text to a `Date` object using the `Date` constructor method  
`Date parse( String s ) throws ParseException`

8

## `loadFromText()`, 2

- Write a second line of code to add the `Date` to the list the method returns.
- Get rid of the line (it's only there to provide a dummy exception throw before you write the real code):  

```
if ( false ) throw new ParseException("",0);
```
- Test the code. Write data out, clear the logger, and see if you can load it back in in text mode.
- Use Notepad to hand write a new logger entry using the date pattern of the previous entries. See if you can read in the modified data. Check it with the Summarize command.

9

## `saveToData()`, 1

- At the heart of the `Date` object is a private `long` data field, the number of milliseconds since 12:00 am 1 Jan 1970.
- `saveToData()` saves log times as `longs` instead of `Strings`. You can retrieve the `long` using the `Date` method `long getTime()`.
- You will need to construct a `DataOutputStream` in order to access methods that can write `longs`. You can't construct one directly from a `File`. You will need to construct a `FileOutputStream` first.

10

## `saveToData( ), 2`

- Use `saveToText( )` as a model for the rest of the method. Remember to catch exceptions.
- Now test. Try writing out the same set of log times as text and data in separate files. Look at both in Notepad. Check the length of both files (you can get an exact byte count by choosing Properties from the right button menu in Explorer).

11

## `loadFromData( ), 1`

- Write a `while` loop to read longs from the `DataInputStream` as long as there are any (check the documentation), to convert each to a `Date` (there is a `Date(long)` constructor), and to add the `Date` to the return list.
- Pay special attention to what happens when you run out of longs in the input stream. How can you exit the `while` loop?
- Compile and test.

12

## `loadFromData(), 2`

- **Make sure you can write log data out in Data mode, clear the logger, and read the same data back in.**
- **Can you create log data in this format using Notepad?**
- **Can you read data written in Data mode back in as text? Can you read Text mode data back in as data? Why?**

13

## `saveToObject(), 1`

- **In this method, we will write out the whole List of Dates as a single object using an `ObjectOutputStream`.**
- **Check the documentation on how to create an `ObjectOutputStream` and how to call the `writeObject()` method. This single call will write the list and everything on it. We will look at how this works in the next lecture.**
- **Compile and test. Write out some log data in Object mode and look at it in Notepad. Check the length of the file. How does it compare to Text and Data mode.**

14

## loadFromObject ( )

- Write the contents of the try/catch block to read the entire log list in using an `ObjectInputStream`. You will have to construct the stream, call `readObject ( )`, and then close the stream.
- Compile and test. Can you read back in a log list that you previously wrote out in Object mode?
- Can you read data in Text or Data mode back in using Object mode?
- What are the comparative advantages of each data mode?

15

## Data Format Scorecard

	text	data	object
data size	2	1	3
viewable, editable	1	3	3
portable	1	2	3
programming ease	3	2	1
error checking	2	3	1

16