

CSSE 220 Day 28

Non-text Files, reading and Writing Objects

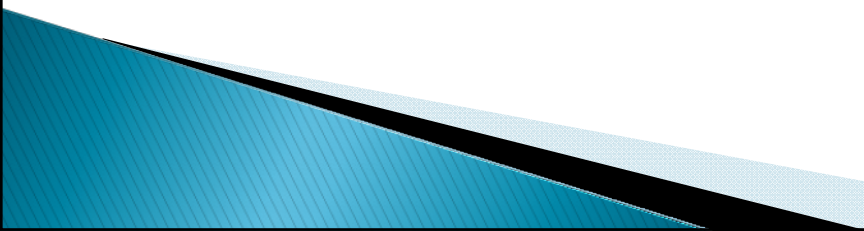
Network IO

Work on Spellchecker Project

CSSE 220 Day 28

- ▶ Everything for the Mini-project is due at the beginning of your class time on Day 30. **No late days** may be used for this one.
- ▶ There will be time in class to work with your team every day. Do not miss it!
- ▶ Writing up and turning in written problems is no longer required. But you should still do them at some point.
- ▶ The Digital Resource Center is looking for a student to do ANGEL support for faculty.
 - See Nancy Bauer in the DRC if you're interested

Course Evaluations

- ▶ I will provide some class time on Thursday for filling out the evaluation forms
 - ▶ I recommend that you wait until then to do them, so you'll be able to comment on the full course, including your project experience.
- 

Project presentation/demonstration

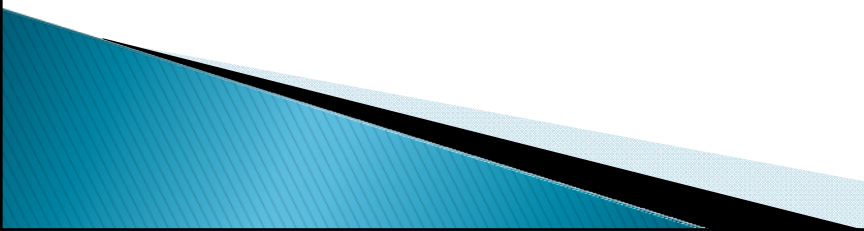
- ▶ Day 30 in class
- ▶ Informal and informational
- ▶ What does your program do? How does it do it
- ▶ Data Structures and algorithms.
- ▶ Intended audience: Your classmates
 - Already know what the project is.
 - Already know Java
 - Already know the data structures we have studied.
- ▶ No more than 7 minutes, including Q&A time.
- ▶ **Just before your presentation, we will randomly choose which of your team members will present, so everyone should be prepared to do it.**
- ▶ **Commit an outline of your presentation to your team repository by 5:00 PM on Tuesday.**

My schedule this week

	11 Monday	12 Tuesday	13 Wednesday	14 Thursday
8 am	CSSE220-01 O269 Anderson, Claude W	CSSE220-01 O269 Anderson, Claude W		CSSE220-01 O269 Anderson, Claude W
9 ⁰⁰			372 Project presentations GM room	
10 ⁰⁰	CSSE220-02 O269 Anderson, Claude W	CSSE220-02 O269 Anderson, Claude W		CSSE220-02 O269 Anderson, Claude W
11 ⁰⁰				
12 pm	craig Zilles UIUC curt's office	372 Scheme Grading; F21 ↻	Sr. ProjectExpo Union	CSSE department CSSE conference room ↻
1 ⁰⁰	CSSE Faculty Lunch with John Georgas			
2 ⁰⁰		PTRC Hadley ↻		PTRC Hadley ↻
3 ⁰⁰				
4 ⁰⁰	Meet with John Georgas	Pray with Matt and Jerry Jerry's Office ↻		
5 ⁰⁰	John Georgas talk	Institute meeting	Special Faculty Meeting E-104	372 Rosie's List; F 210 ↻

- ▶ As always, you can find my up-to-date schedule online.

Questions from students

- ▶ Spellchecker
 - ▶ Sorting
 - ▶ Input and output
 - ▶ Anything else
- 

**THE DEPARTMENT OF COMPUTER SCIENCE
& SOFTWARE ENGINEERING**

**INVITES YOU TO THE
FACULTY CANDIDATE TALK**

**JOHN GEORGAS
UNIVERSITY OF CALIFORNIA, IRVINE**

**SUPPORTING ARCHITECTURE- AND POLICY-BASED
SELF-ADAPTIVE SOFTWARE SYSTEMS
MONDAY FEBRUARY 11, 2008**

4:30 P.M. O-269

Please
stay
afterward
to talk
informally
with John.

Today's Agenda

- ▶ Random access files and serialization
 - ▶ Networking intro
 - ▶ Work on Spellchecker
- 

Text Files vs Binary files

```
public static void main(String[] args) throws IOException{
    int [] nums = new int [20];
    for (int i=0; i<nums.length; i++) {
        nums[i] = (int)(Math.random()*Integer.MAX_VALUE);
    }
    PrintWriter pw = new PrintWriter(
        new FileOutputStream("text.txt"));
    DataOutputStream os = new DataOutputStream(
        new FileOutputStream("bin.bin"));

    for (int n : nums) {
        pw.print(n + " ");
        os.writeInt(n);
    }
    pw.println();
    pw.close();
    os.close();
}
```

What is the difference between the effects of these two statements?

```
>ls -l bin.bin text.txt
a-----      80  8-Feb-108 13:50 bin.bin
a-----     211  8-Feb-108 13:50 text.txt
```

UNIX output format is more compact than MSDOS.

Random Access Files

Streams provide easy sequential access to a file, but sometimes you want to have random access; for example a database program certainly needs to be able to go directly to a particular location in the file.

```
import java.io.*;
public class RandomAccess {
    public static void main(String [] args) {
        try {
            RandomAccessFile raf = new RandomAccessFile("random.dat", "rw");
            for (int i=0; i<10; i++)
                raf.writeInt(i);
            raf.seek(20);
            int number = raf.readInt();
            System.out.println("The number starting at byte 20 is " + number);
            raf.seek(4);
            number = raf.readInt();
            System.out.println("The number starting at byte 4 is " + number);
            raf.seek(5);
            number = raf.readInt();
            System.out.println("The number starting at byte 5 is " + number);

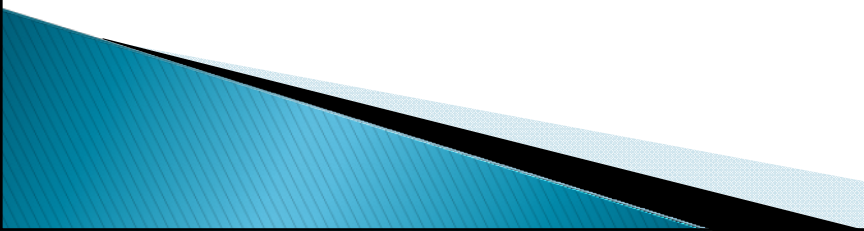
            raf.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

writeInt ?

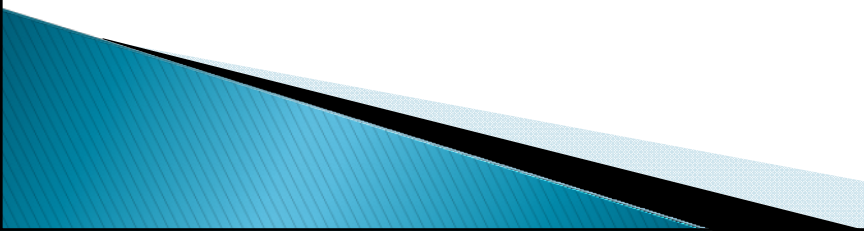
Note that we are reading and writing numbers in their internal (binary) representation, not in their text (human-readable) representation.

This example is adapted from Art Gittleman, *Advanced Java:Internet Programming*, page 16

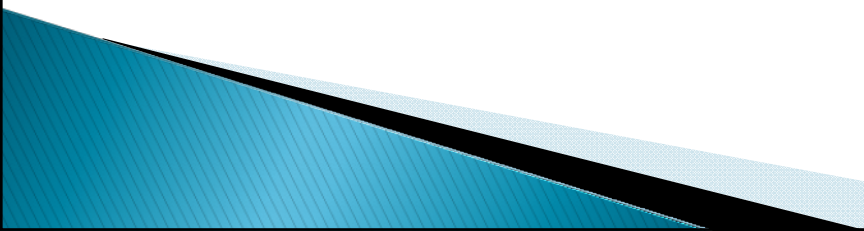
Reading and Writing Objects

- ▶ We'd like to be able to write objects to a file, then read them back in later.
 - ▶ Java (transparently to the user) writes type information along with the data.
 - ▶ Reading the object in will recover its type information.
- 

Issues with reading/writing Objects

- ▶ Objects can contain references to other objects.
 - Writing out the actual reference (a memory address) would be meaningless when we try to read it back in.
 - ▶ Several objects might have references to the same object.
 - We do not want to write out several copies of that object to the file.
 - If we did, we might read them back in as if they were different objects.
- 

Solution: Object Serialization

- ▶ The objects that we write/read must implement the **Serializable** interface (which has no methods).
 - ▶ Objects are written to an `ObjectOutputStream`.
 - ▶ An example should help you see how it works.
- 

Example: 1. Serializable classes

```
class Person implements Serializable{
    private String name;
    public Person (String name) {
        this.name=name; }
}
```

```
class Account implements Serializable {
    private Person holder;
    private double balance;
    public Account(Person p, double amount) {
        holder=p;
        balance=amount;
    }
}
```

Note that an Account
HAS-A Person

```
class SavingsAccount extends Account implements Serializable {
    private double rate;
    public SavingsAccount(Person p, double amount, double r) {
        super(p,amount);
        rate=r;
    }
}
```

Example: 2. Definitions and Output

```
public static void main(String [] args) {  
    try {  
        Person fred = new Person("Fred");  
        Account general = new Account(fred, 110.0);  
        Account savings = new SavingsAccount(fred, 500.0, 6.0);  
  
        ObjectOutputStream oos = new ObjectOutputStream(  
            new FileOutputStream("Objects.dat"));  
        oos.writeObject(general);  
        oos.writeObject(savings);  
        oos.close();  
    }  
}
```

- ▶ In addition to `writeObject()`, the `ObjectOutputStream` class provides methods for writing primitives, such as `writeDouble()` and `writeInt()`. `writeObject()` calls these when needed.

Example: 3. Input Serialized Objects

```
ObjectInputStream ois =  
    new ObjectInputStream(  
        new FileInputStream("Objects.dat"));  
Account aGeneral = (Account)ois.readObject();  
Account aSavings = (Account)ois.readObject();
```

- ▶ We must read the objects in the same order as they were written.
- ▶ Both objects that are read are assigned to variables of the type **Account**, even though one should have been written out as a **SavingsAccount**.
- ▶ We will check to make sure it was read correctly.

Example: 4. Check the Objects

```
if (aGeneral instanceof SavingsAccount)
    System.out.println("aGeneral is a SavingsAccount");
else if (aGeneral instanceof Account)
    System.out.println("aGeneral is an Account");
if (aSavings instanceof SavingsAccount)
    System.out.println("aSavings is a SavingsAccount");
else if (aSavings instanceof Account)
    System.out.println("aSavings is an Account");
if (aGeneral.holder == aSavings.holder)
    System.out.println("The account holder, fred, is shared");
else
    System.out.println("Account holder, fred, was duplicated");
ois.close();
catch (IOException ioe) {
    ioe.printStackTrace();
catch (ClassNotFoundException cnfe) {
    cnfe.printStackTrace();
```

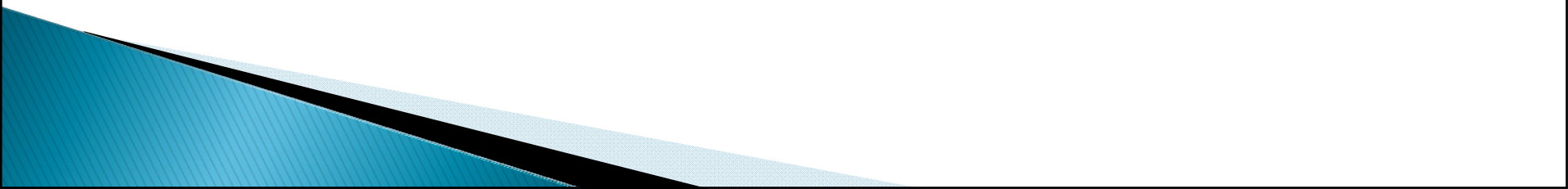
Output:

```
aGeneral is an Account
aSavings is a SavingsAccount
The account holder, fred, is shared
```

What if the Input/Output is from/to a Network Server?

- ▶ Network programming in java

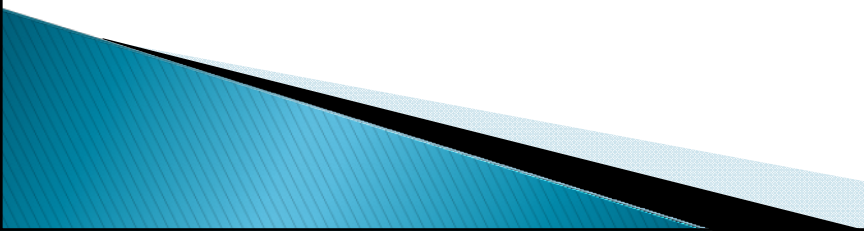
Network programming

- ▶ Let's start with what you know.
 - ▶ What are some terms, concepts, and issues associated with network communication and network programming?
- 

Network programming

- ▶ Most network programs involve a **server** program and one or more **client** programs.
- ▶ When a server is started, it is associated with an Internet **port** number. Port numbers below 1024 are generally reserved for system services; user-written services use higher port numbers.

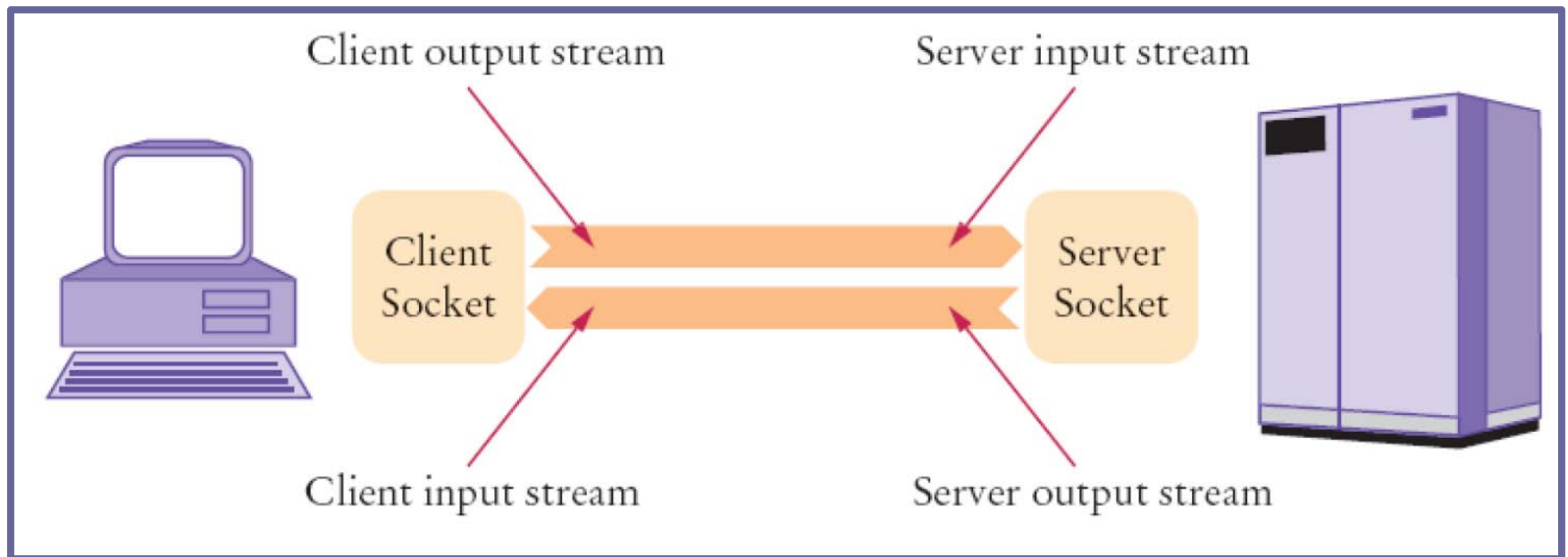
Network programming

- ▶ Programs typically connect *via* a socket, and communicate using an agreed-upon protocol.
 - ▶ If you randomly choose a server program and a client program, they probably can't communicate because they use different protocols.
 - ▶ We can use a standard protocol (such as TELNET, HTTP or FTP) or make up our own.
- 

Socket

- ▶ A socket is the standard intermediate-level model of a client-server connection
- ▶ The client and server each provide a socket, which is "half of the connection"
 - Examples: AC connection socket, DC connection socket, Monitor connection socket
- ▶ After being established on a port, the server creates its end of the socket and waits for a client to connect (*via accept* command)
- ▶ Many APIs, including JDK, provide higher-level tools, such as URLConnection objects

Communicating over a socket



Note: This slide and several subsequent slides, along with the corresponding code, were adapted from *Big Java* by Cay Horstmann

Host Addresses

- ▶ **Ethernet address** (MAC address)
 - 12 hexadecimal numbers
 - used mainly for assigning IP address. One of mine is **00-1B-77-47-DE-DF**
- ▶ **IP address**
 - 4 numbers (in range 0–255) separated by periods
 - As I am writing this, mine (via VPN connection) is **137.112.248.114**
All RHIT addresses begin with 137.112
- ▶ **Domain-name address**
 - addiator.rose-hulman.edu
 - www.rose-hulman.edu
 - A *name-server* (DNS) translates from domain-name addresses to IP addresses
 - Usually the name server's work is transparent to the user

Anatomy of an HTTP URL

- ▶ URL stands for **Uniform Resource Locator**
- ▶ **HTTP://www.rose-hulman.edu:80/class/csse**
- ▶ **HTTP**: HyperText Transfer Protocol. Other protocols are possible, such as FTP, MAILTO, FILE.
- ▶ **www.rose-hulman.edu** The address (can also be specified as an IP address: 137.112.255.80).
- ▶ **80** The internet port number. A server establishes port numbers for its network services so that clients can locate them.
80 is the default for HTTP
21 for FTP
- ▶ **class/csse** Directory and/or file information.
- ▶ Many browsers attempt to fill in missing parts.

The OSI Reference Model

