Principles of Software Construction: Objects, Design and Concurrency

Stream I/O in Java

15-214

toad

Fall 2012

Jonathan Aldrich **Charlie Garrod**

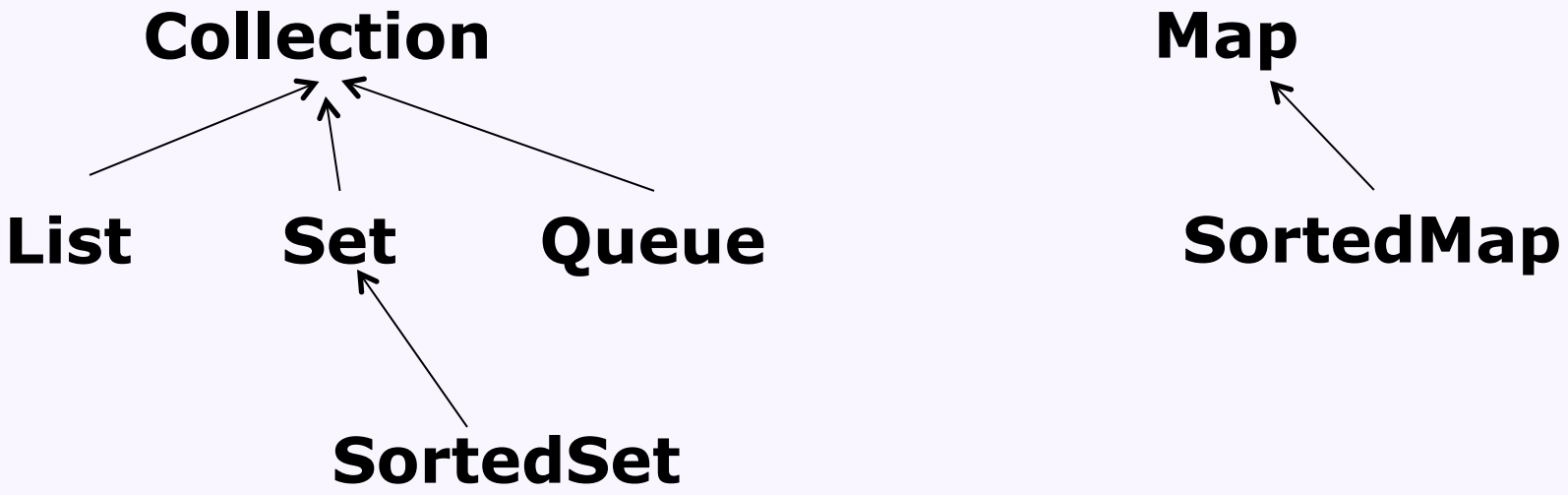
School of Computer Science
institute for SOFTWARE RESEARCH

Administrivia

- Homework 6 team sign-ups due tonight
  - You may not use late-days for the sign-up process
  - See the Piazza note for details

Last time: The Java Collections Framework

- Interfaces (in java.util)



- Default Implementations
  - ArrayList, LinkedList, HashSet, TreeSet, PriorityQueue, HashMap, TreeMap, LinkedHashSet, LinkedHashMap, …
- Algorithms
  - min, max, sort, reverse, binarySearch, shuffle, rotate, …

A question for you:

- Why is this the Java Collections *Framework*?
  (…and not just the Java Collections Standard Library?)

A question for you:

- Why is this the Java Collections *Framework*?
  (…and not just the Java Collections Standard Library?)
  (Where is the extensibility?)
- One answer:
  - ArrayList, LinkedList, HashSet, etc. are merely default implementations
    - There are other specialty implementations
    - You can write your own

Today: Stream I/O and Networking in Java

- Basic I/O in Java
- Distributed systems
- Networking in Java
  - Communication via network sockets
  - Java RMI

System.out is a java.io.PrintStream

- `java.io.PrintStream`: Allows you to conveniently print common types of data

```
void close();
void flush();
void print(String s);
void print(int i);
void print(boolean b);
void print(Object o);
…
void println(String s);
void println(int i);
void println(boolean b);
void println(Object o);
…
```

The fundamental I/O abstraction: a stream of data

- `java.io.InputStream`

```
void           close();
abstract int   read();
int            read(byte[] b);
```

- `java.io.OutputStream`

```
void           close();
void           flush();
abstract void  write(int b);
void           write(byte[] b);
```

- Aside: If you have an `OutputStream` you can construct a `PrintStream`:

```
PrintStream(OutputStream out);
PrintStream(File file);
PrintStream(String filename);
…
```

We typically want structured input, too

- e.g., `java.util.Scanner`

```
Scanner(InputStream source);
Scanner(File source);
void    close();
boolean hasNextInt();
int     nextInt();
boolean hasNextDouble();
double  nextDouble();
boolean hasNextLine();
String  nextLine();
boolean hasNext(Pattern p);
String  next(Pattern p);
…
```

See the FileExample.java demo

- Note the output format

To read and write arbitrary objects

- Your object must implement the `java.io.Serializable` interface
  - Methods: none!
  - If all of your data fields are themselves `Serializable`, Java can automatically serialize your class
    - If not, will get runtime `NotSerializableException`
- See QABean.java and FileObjectExample.java

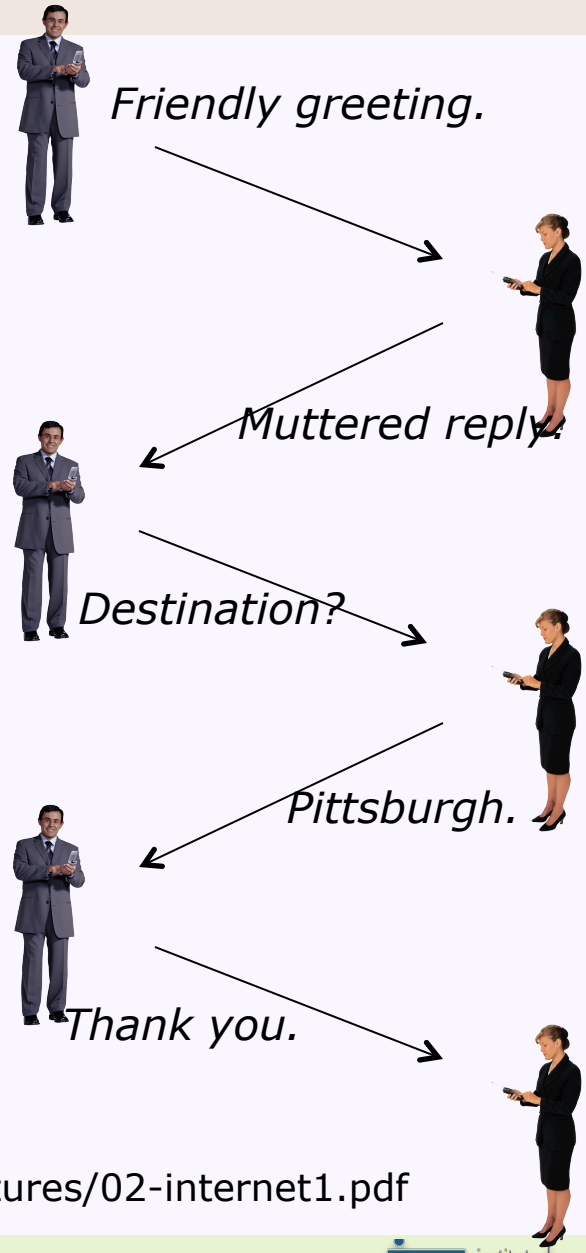
Distributed systems

- Multiple system components (computers) communicating via some medium (the network)
- Challenges:
  - Heterogeneity
  - Scale
  - Geography
  - Security
  - Concurrency
  - Failures

(courtesy of http://www.cs.cmu.edu/~dga/15-440/F12/lectures/02-internet1.pdf

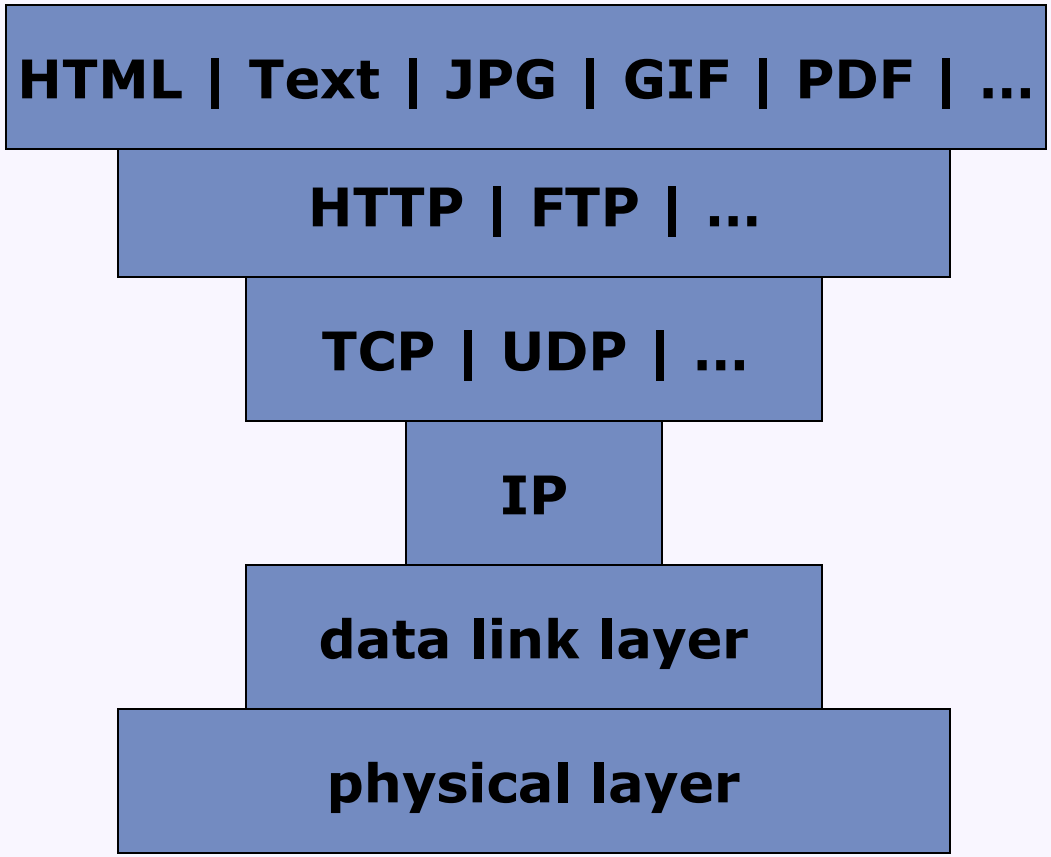
Communication protocols

- Agreement between parties for how communication should take place
  - e.g., buying an airline ticket through a travel agent



(courtesy of http://www.cs.cmu.edu/~dga/15-440/F12/lectures/02-internet1.pdf

Abstractions of a network connection



Packet-oriented and stream-oriented connections

- UDP: User Datagram Protocol
  - Unreliable, discrete packets of data
- TCP: Transmission Control Protocol
  - Reliable data stream

Internet addresses and sockets

- For IP version 4 (IPv4) host address is a 4-byte number
  - e.g. 127.0.0.1
  - Hostnames mapped to host IP addresses via DNS
  - ~4 billion distinct addresses
- Port is a 16-bit number (0-65535)
  - e.g. 80
  - Assigned conventionally
- In Java:
  - `java.net.InetAddress`
  - `java.net.Inet4Address`
  - `java.net.Inet6Address`
  - `java.net.Socket`
  - `java.net.InetSocket`

Networking in Java

- The java.net.InetAddress:

```
static InetAddress getByName(String host);
static InetAddress getByAddress(byte[] b);
static InetAddress getLocalHost();
```

- The java.net.Socket:

```
Socket(InetAddress addr, int port);
boolean       isConnected();
boolean       isClosed();
void          close();
InputStream   getInputStream();
OutputStream  getOutputStream();
```

- The java.net.ServerSocket:

```
ServerSocket(int port);
Socket        accept();
void          close();
…
```

A simple Sockets demo

- TextSocketClient.java
- TextSocketServer.java
- TransferThread.java

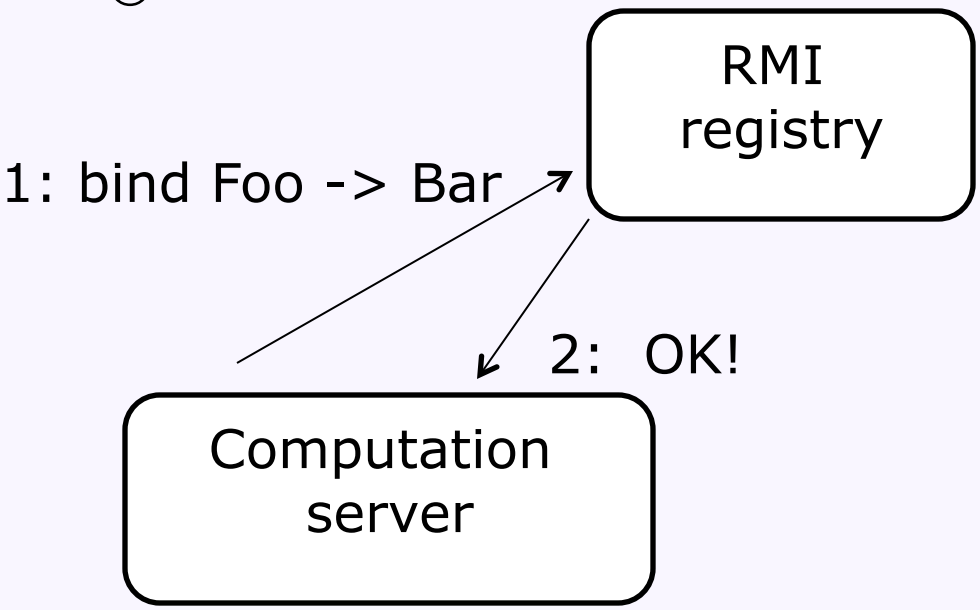
What do you want to do with your distributed system today?

Higher levels of abstraction

- Application-level communication protocols
- Frameworks for simple distributed computation
  - Remote Procedure Call (RPC)
  - Today: Java Remote Method Invocation (RMI)
- Complex computational frameworks
  - e.g., distributed map-reduce

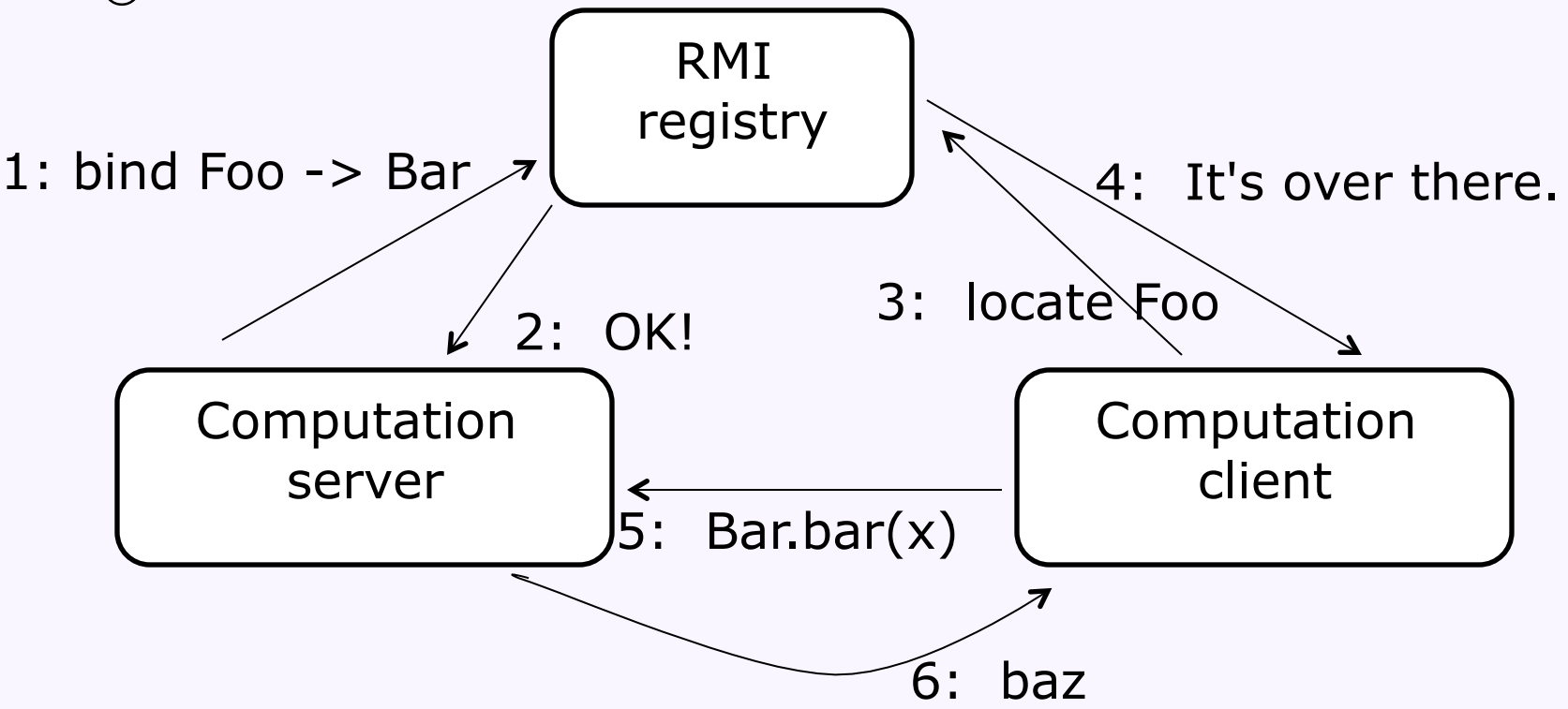
Java Remote Method Invocation (RMI)

- Abstracts away the location of the computation
  - Use just like a method call
  - Automatic communication of arguments and return values
- Java-specific
  - ☹



Java Remote Method Invocation (RMI)

- Abstracts away the location of the computation
  - Use just like a method call
  - Automatic communication of arguments and return values
- Java-specific
  - ☹



Creating an RMI server

- Must implement `java.rmi.Remote`
  - No required methods, just a marker interface
  - All methods must throw `java.rmi.RemoteException`
- Set a SecurityManager to allow RMI
  - e.g., `java.rmi.RMISecurityManager`
- Create a server stub
  - `java.rmi.server.UnicastRemoteObject`
    - `Remote exportObject(Remote obj, int port)`
- Bind your stub to a name at some RMI registry
  - `java.rmi.registry.LocateRegistry`
    - `Registry getRegistry(String host)`
  - `java.rmi.registry.Registry`
    - `void bind(String name, Remote obj)`
    - `void rebind(String name, Remote obj)`

Creating an RMI client

- Set a SecurityManager that allows RMI
- Look up client stub using name in RMI registry
  - `java.rmi.registry.LocateRegistry`
    - `Registry getRegistry(String host)`
  - `java.rmi.registry.Registry`
    - `Remote lookup(String name)`
- Use the client as if it were a local object
- See:
  - Compute.java
  - Operation.java
  - AddOp.java
  - ComputeServer.java
  - ComputeClient.java

RMI: dealing with failure

- Problem: the network is unreliable
- `java.rmi.RemoteException`
  - Did the compute server receive my last request?
  - Is the compute server running?
  - What happens if I send the same request again?
    - How many times did the method run?

Next week:

- Concurrency in Java