Misc

• Homework #1 is past due
• Midterm Evaluations
Lecture 6

Asserts, Initialization, super, finalize()
Network Programming -
Sockets & RMI
Assert

- If the value is **false** in runtime and assertions are turned on, throws an exception that halts the program, unless caught
- Can be turned on/off at execution time by a command line parameter to the JVM
  - Java \(-ea\) MyApplication
- Can be turned on/off at a package/class level
Enabling in Netbeans

- Part of Java 1.4
- Requires a switch for the compiler
  - "-source 1.4"  //will be auto set by netbeans
- Go to netbeans help, look for assertions and follow the directions.
- Turn on/off for execution in the tool
  - Tools >> Options >> Debugging and Executing >> Execution Types >> External Execution >> Expert >> Enable Asserts
Sample Assertion Code

```java
public void willAssertIfPassedFalse(boolean valueToTest)
{
    System.out.println("=====================================================");
    System.out.println("This is the willAssertIfPassedFalse() method in the " + this.getClass() + "\n\n    System.out.println("valueToTest = "+ valueToTest);
    //assert(valueToTest); this is one form
    assert valueToTest : "this is a custom statement in an assert. valueToTest is false";
    System.out.println("=====================================================");
}

public static void main(String[] args)
{
    AssertionDemo myAssertionDemo = new AssertionDemo();

    myAssertionDemo.willAssertIfPassedFalse(true);
    myAssertionDemo.willAssertIfPassedFalse(false);
}
```

Use of Assertions

• Not to check parameters of public methods
  – use if statements, generally

• Showstopper conditions
  – Things that should never happen in the normal execution
  – Turn on in testing and off in deployment

• Often in case/switch/default
Variable Initialization vs. Scope

• Static – automatic by java to null, 0, 0.0, …
• Instance – automatic by java to null, 0, 0.0
• Local – you have to set them (somewhere) or get errors/exceptions

```java
Public class DemoThis {
    static int MyInt;               //permanent – heap allocated
    static float MyFloat;
    Static StringBuffer MyStringBuffer;

    int myInstanceInt;           //life of the object, allocated at constructor time – heap allocated
    Float                     myInstanceFloat;
    StringBuffer myInstanceStringBuffer;

    Public void someMethod(int myArgInt)    //pushed on the stack
    {
        int myLocalInt = 100;  //these are temporary – the life of the method invocation - stack
        float myLocalFloat = 0.0;
        StringBuffer myLocalStringBuffer = null;
    }
}
```
Super

- Can be used to refer to the parent class or parent instance methods/vars
- Example:

  ```java
  Public Square()
  {
      super(); //calls the super constructor
  }
  ```

- But isn’t the parent constructor automatically called for me in my constructors? – yes, the 0 parameter one
Super

• Example:

```java
public void someMethod()
{
    int sum = super.sum;
}
```

• Example:

```java
super(5); //invokes a param constructor
//java no longer invokes the default
//must be the first statement
```
finalize() method

- Called just before garbage collection
- No guarantee of getting garbage collected, though!
- Don’t get garbage collected until 0 references to an object
- finalize() of super not automatically called
Arrays and the Object Hierarchy

- Arrays are a parallel hierarchy to the class type they hold.
- Use `instanceof` to check the type of if moving an element from one array to another e.g. `Square[]` to `Polygon[]`
What is a socket

- A layer of code that runs on top of a lower level network protocol like IP
- TCP, UDP and Multicast Sockets
- A connection between two parties via a layer on top of IP
Client & Server Applications communicate via the protocols

Internet/WWW Protocol Layers

UDP (Unreliable)  TCP (Reliable)

IP

Ethernet 802  SLIP/PPP

Cat 1-5  FDDI  Frame Relay  ATM  ISDN  ADSL  X.25
IP

- Internetworking Protocol – 1982?
- Runs on top of Ethernet (IEEE802) protocol and others (ATM, FDDI, X.25)
- Provides sending packets between addresses
- “Fire and forget” protocol
- IPv4 addresses are 4 numbers separated by dots
  - 10.255.0.128 (all numbers are between 0-255 or 0x00 - 0xFF hexadecimal or a 32 bit binary number. It takes 4 bytes to hold the address.)
  - Means $2^{32} = 4,294,967,296 - 1 \approx 4.29 \times 10^9$ possible addresses!
  - Like “phone numbers” for internet hosts
IP

• Connectionless (like sending a letter not like a phone call)
• Addressing and route selection
• Packet assembly and disassembly
• Header
  – Version
  – Source Address – 4 bytes
  – Destination Address – 4 bytes
  – Checksum
  – TTL (Time To Live) – number of router hops allowed
  – Total Length
  – Payload – between 1 – 64K bytes long
IP Addresses

• **Static**
  – Never changes – can be manually in OS setup.
  – Assigned to the MAC address NIC

• **Dynamic**
  – DHCP (Dynamic Host Configuration Protocol)
  – pool of addresses - leased
  – Expire then you get new one
  – Efficient and flexible
TCP

• Transmission Control Protocol
• Connection oriented
• Guaranteed packet delivery – error free, in sequence sent, no dupes,
• Runs on top of IP
• Slower than UDP
• Each IP Address also can have up to 65536 TCP Ports
  – Like phone extensions on a phone number.
TCP Socket

• IP Address + TCP Port Numbers of Client and Server
• PTP – Point to Point – like a phone call
• Socket connection - handshaking C/S Server machines. Establishes Packets sent get acknowledged by the receiver.
• Congestion control
• Was system dependant
• BSD Sockets machine independent
TCP Packet

- Source TCP Port – 2 bytes
  - Address is in the IP Header
- Target TCP Port - 2 bytes
  - Address is in the IP Header
- Sequence Number – 4 bytes = 0 – \(2^{32}-1\)
- Acknowledgement Number
- Window – size of the senders buffer
-Checksum – 2 bytes
- Control Bits
- Packet Size is in the IP header
- Payload

IP Packet

TCP Packet
Well Known TCP Ports (0-1024)

- 80  HTTP
- 443  SSL
- 20 & 21  FTP (20 control, 21 data)
- 25  SMTP
- 110  POP3
- 23  TelNet
- 161  SNMP
- 7  Echo
- 53  DNS
- others
Socket communication Using TCP

Initiator Socket App

Initiator Socket App

Acceptor Socket App

TCP (Reliable)

IP

Ethernet 802

Cat 1-5
If you don’t have a network card you CAN have a permanent IP address on your machine.
adding a loopback adapter
add a new device
select 'no'
select other devices
microsoft & loopback adapter
click next
finished installing the loopback adapter
you should have a new entry in your networking and dialup
properties
properties for TCP/IP
set it to a convenient address like 100.100.100.100
Getting the IP address of a machine

- on windows create a DOS/command prompt
- run ipconfig.exe or winipconfig.exe
- should be on your machine somewhere you may need to search for it
• Questions
• Comments

• 7.5
What is localhost?

- Is a short cut that says - use this machine’s default IP address
- You could use 127.0.0.1 which is the loopback address for any machine also but it means you can’t talk to any other boxes
TCP demo

- TCPClientGUI
- TCPEchoTextServerDemo
<<An Exception was thrown on the client>> : java.net.UnknownHostException: 100.100.100.300
TCP Client Application

Server URL: 100.100.100.101
Port: 8080

Connect

Text To Send

<<An Exception was thrown on the client>> : java.net.ConnectException: Connection refused: no further information

SendText

Text Returned From Server
run the TCPEchoTextServerDemo

• Output:

<<socket factory created>>
<<server host address = 0.0.0.0>>
<<server host port = 8081>>
<<server SO timeout = 0>>
<<socket factory returned a new connection : connection number 1>>
Sending a string to the server

- Server output:

  [1] String Received = This is a string send by my TCPClientGUI application
TCPClientGUI

• Creates a stream (TCP) Socket
• The constructor creates it and attempts to connect with the server on the specified port and IP address.
• Gets a PrintWriter to send text strings instead of just stream of bytes
• Gets a BufferedReader to get back strings instead of just bytes.
TCPEchoTextServerDemo

• **Creates a ServerSocket** (a factory that returns a new socket for every client that connects)

• create a thread to run the processing for that socket on

• start the thread

• listen for more client connections
TCPEchoTextServerDemo

- Gets a PrintWriter to send text strings instead of just stream of bytes
- Gets a BufferedReader to get back strings instead of just bytes.
- Sends a message to the client saying “Hello to my client”
- Loops waiting for a string from the reader and then just appends some text and returns it through the writer.
Three clients are talking to a single multithreaded server application.
output from the server

<<socket factory created>>
<<server host address = 0.0.0.0>>
<<server host port = 8081>>
<<server SO timeout = 0>>
<<socket factory returned a new connection : connection number 1>>
<<socket factory returned a new connection : connection number 2>>
<<socket factory returned a new connection : connection number 3>>

[3] String Received = This is text from client three
[2] String Received = This is from client number Two the best client, I'm told
Multithreaded servers

• Efficient to a certain extent but …
• If the clients aren’t talking much it wastes resources
• So, maybe you should limit the number of threads you create and just reuse them
• That’s thread pooling
• And maybe your clients should connect, do their work, disconnect, then later reconnect, so some work, … that’s HTTP’s strategy for scaling.
TCP sockets are streams of bytes

• You can send almost any type of object not just bytes
• Primitives int, long, …
• String, and other java objects
• HTTP uses TCP sockets
TCP parameters

- **SO_TIMEOUT** – amount of time to block waiting, if exceeded throws a `InterruptedIOException`
- **TCP_NODELAY** – sends packets immediately no matter how small
- **SO_LINGER** – should it wait after a close to try and send unsent data?
- **TCP_KEEPALIVE** – detects lost clients by sending a packet every two hours if no packets sent by client. Not part of TCP.
- `shutdownOutput()` - signals the other end that no more data will be sent
- `shutdownInput()` - signals the other end its not accepting any new data
Initiator & Acceptor

• The code that creates a socket is the initiator of the socket
• The code that waits and accept() ‘s the socket is the acceptor
• Often incorrectly termed client and server
• Both sides of the socket connection can send packets without the other sending one first, unlike the client-server paradigm
Initiator code

```java
mySocket = new Socket(serverURLText.getText(),
              Integer.parseInt(serverTCPPortText.getText()));
myInputStream = mySocket.getInputStream();
myOutputStream = mySocket.getOutputStream();
myBufferedReader = new BufferedReader(new
                          InputStreamReader(myInputStream));
myPrintWriter = new PrintWriter(myOutputStream,true);

Thread.currentThread().sleep(1000); //pause to let the server
    //socket get running
if (myBufferedReader.ready()) //see if any message from the
    //server to begin with
    textReturnedTextarea.setText(myBufferedReader.readLine());
```
public static void main(String[] args) {
    ServerSocket mySocketFactory = null;
    int numberOfActiveConnections = 0;

    try {
        if (args.length == 0) // no commandline args
            mySocketFactory = new ServerSocket(8081);
        else {
            mySocketFactory = new ServerSocket(Integer.parseInt(args[0]));
        }
    }
}
Acceptor code (2)

TCPEchoTextServerDemo CurrentTCPEchoTextServerDemoInstance;
Socket socketReturnedFromFactory = mySocketFactory.accept();
numberOfActiveConnections++;

System.out.println("<<socket factory returned a new connection:
    connection number " +
    Integer.toString(numberOfActiveConnections) + ">>");

currentTCPEchoTextServerDemoInstance =
    new TCPEchoTextServerDemo( socketReturnedFromFactory,
    numberOfActiveConnections );
//this gets a socket to listen on
//now start a new thread for that socket's processing
Acceptor code (3)

public TCPEchoTextServerDemo(Socket socketToRunOnThread,  
   int connectionID)  
   throws Exception
{
   myConnectionID=connectionID;  
   mySocket = socketToRunOnThread;  
   myThreadIRunOn = new Thread( this );  
   myThreadIRunOn.start();
}

• Constructor creates a new thread to run itself on
• Gets a socket from the instantiator ( the main() )
UDP

- User Datagram Protocol
- Not connection oriented
- “Fire and forget” IP
- No guarantee – lost packets, dups, and out of sequence ok
- Faster than TCP – lower overhead
UDP Packet

- Inside the payload of an IP packet
- Source & Destination Port
  - Addresses are in the IP Header
- Checksum – 2 bytes
- Message Length is in IP Header
Socket communication Using UDP

Sender Socket App

Listener Socket App

SenderListener Socket App

UDP (Fire and Forget)

IP

Ethernet 802

Cat 1-5
Sending and listening on the same addresses and port

• This is a single GUI talking to itself
two instances talking on the same machine (two ports)
A simple UDP chat application

Hey you over there, I'm sending text from the LEFT client.

This is my response from the RIGHT client sending a message to you!

Hey you over there, I'm sending text from the LEFT client.

This is my response from the RIGHT client sending a message to you!
Using same port but different IP addresses

Sent from IP 100.100.100.101 this is two ips on the same port

Sent back from 169.254.25.129 to socket on IP 100.100.100.101 port 8082

Sent from IP 100.100.100.101 this is two ips on the same port

Sent back from 169.254.25.129 to socket on IP 100.100.100.101 port 8082
Datagram Socket

- No initiator and acceptor
- You get a socket and can send a datagram packet to an IP and Port
- If nobody’s listening it’s ok! It gets thrown away.
- You can just listen and not send, too!
Sending without listening

This SHOULD not get received as the other client isn't listening

This should get received because the other client is listening

Copyright Poelman & Associates, Inc. 2003 - All Rights Reserved
• Questions

• Comments

• 8.0
RMI

• Remote Method Invocation
• The ability to invoke a method on an object that may be in a different process, JVM or machine as though it was a local object
Using RMI

• Create a interface that implements the methods to be remoted and implement the Remote interface
  - public interface DemoRMIServerRemote extends Remote

• Create class to be remotely invoked that implements the interface
  - public class DemoRMIServerImpl extends java.rmi.server.UnicastRemoteObject implements DemoRMIServerRemote

• Run the rmic compiler on the class it generates stubs and skeletons
  - DemoRMIServerImpl_Stub.class
  - DemoRMIServerImpl_Skel.class
Using RMI (2)

- **Run rmiregistry**
- In the class to invoke remotely register the instance with the `rmiregistry` executable
- The client looks up the remote class in the `rmiregistry`
- The client invokes the stub, the stub marshalled the parameters and sends them to the skeleton
- The skeleton invokes the method on the running remote server gets the return values, marshalls them and returns them to the stub
- The stub unmarshalls the values and returns them to the client
rmic (rmi compiler)

• creates stubs and skeletons for your class
• open a command prompt
• cd to Lectures
• Make sure to SET YOUR CLASSPATH to point to those files directly! or rmic won’t work
• type rmic Lecture6.DemoRMIServerImpl
• creates files for you in the Lecture6 sub directory
rmic and generated class files
running the RMI registry

- Starts a process that listens on port 1099 on your machine
A client making a local method call
A client making a remote method invocation

Box 1 – JVM 1

Box 2 – JVM 2

TCP

Client

Remote Server

rmiregistry

rmic generates stub and skeleton
RMI sends all objects by value!

- Objects are copied and serialized
- Only return objects will contain changes made in the remote object
- Only methods get remoted NOT attributes (see the interface definition) – you need to provide setter/getters)
- Except for objects that have remote references
• Questions
• Comments

• 8.5
Summary

- TCP (Stream) Socket is connection between the initiator and acceptor
- TCP acceptor uses a socket factory to support multiple initiators and a single acceptor
- UDP (Datagram) socket is connectionless (fire and forget)
- RMI allow invoking methods on an object remotely (over TCP)
- Requires creating “extra” interface and generating stubs and skeletons via \texttt{rmic}
- Look up a remote object via the \texttt{rmiregistry} and invoke it through the stub returned by the \texttt{rmiregistry}.
- RMI can send object that contain other objects, etc. A complete graph of objects can be marshalled (serialized) and unmarshalled (deserialized).
Homework #2

- Using Sockets
- Using RMI
- See the website
Resources

- ..\Java Platform Documentation\docs\guide\rmi\index.html