asynchronous interactions
learning objectives

- learn what asynchronous interactions are
- learn about asynchronous methods in java
- learn about tcp/udp sockets and web sockets
- learn about message-oriented middleware and jms*
What is an asynchronous interaction?

- No blocking of the client until the server is done.
- No polling by the client when a result is expected from the server.

![Diagram of synchronous, polling, and asynchronous interactions]

Asynchronous interactions allow to achieve time decoupling.
asynchronous methods

they rely on the notion of future objects

these objects are also called promises

a session bean can implement asynchronous methods

the container returns the control to the client before the method is actually invoked in background

the client can try to get but might be blocked if it is not ready yet
asynchronous methods

an asynchronous method must return `void` or a `Future<V>` object

if it returns `void`, it cannot declare exceptions

the client can use the `Future<V>` object to retrieve the actual result or to cancel the invocation
asynchronous methods

@Remote
public interface PortfolioRemote {
    public Future<Double> computeValue();
}

Future<Double> value = myPortfolio.computeValue();
System.out.println("Portfolio is worth $" + value.get());

Future<Double> value = myPortfolio.computeValue();
try {
    System.out.println("Portfolio is worth $" + value.get(5, TimeUnit.SECONDS));
} catch (TimeoutException ex) {
    value.cancel(true);
    System.err.println("Timeout: operation was cancelled");
}

@Asynchronous
public Future<Double> computeValue() {
    if (context.wasCancelCalled()) {
        System.err.println("Call to computeValue() was cancelled");
        return null;
    }
    double value = ...;  // Processor-intensive computation
    return new AsyncResult<Double>(value);
}
asynchronous messaging
using sockets

distributed application

application
presentation
session
transport
network
data link

logical peer-to-peer link

physical link

the OSI* model

*Open Systems Interconnection

- stream oriented
- reliable channels
- FIFO ordering

Transmission control protocol
Internet protocol

- packet oriented
- best-effort routing
- error detection
- datagram fragmentation
an IP address is used by the IP protocol to address computers and routers. An IP V4 address consists of 32-bits (4 bytes) and is often written in dotted decimal format, e.g., 130.223.171.8.

<table>
<thead>
<tr>
<th>Class</th>
<th>First byte</th>
<th>Networks</th>
<th>Hosts</th>
<th>Address format</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1→126</td>
<td>$2^7 - 2$</td>
<td>$2^{24} - 2 = 16'777'214$</td>
<td>net id</td>
</tr>
<tr>
<td>B</td>
<td>128→191</td>
<td>$2^{14}$</td>
<td>$2^{16} - 2 = 65'534$</td>
<td>net id</td>
</tr>
<tr>
<td>C</td>
<td>192→223</td>
<td>$2^{21}$</td>
<td>$2^8 - 2 = 254$</td>
<td>net id</td>
</tr>
<tr>
<td>D</td>
<td>224→239</td>
<td></td>
<td></td>
<td>multicast</td>
</tr>
<tr>
<td>E</td>
<td>240→247</td>
<td></td>
<td></td>
<td>reserved</td>
</tr>
</tbody>
</table>
asynchronous messaging
using sockets

internet protocol

ip v4 address

<table>
<thead>
<tr>
<th>Class</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>\textcolor{red}{0NNNNNNNN} \textcolor{green}{.HHHHHHHHH.HHHHHHHH.HHHHHHHH}</td>
</tr>
<tr>
<td>B</td>
<td>\textcolor{red}{10NNNNNN} \textcolor{green}{.NNNNNNNN.HHHHHHHH.HHHHHHHH}</td>
</tr>
<tr>
<td>C</td>
<td>\textcolor{red}{110NNNNN} \textcolor{green}{.NNNNNNNN.NNNNNNNN.HHHHHHHH}</td>
</tr>
<tr>
<td>D</td>
<td>\textcolor{red}{1110MMMM} \textcolor{green}{.MMMMMMMM.MMMMMMMM.MMMMMMMM}</td>
</tr>
<tr>
<td>E</td>
<td>\textcolor{red}{1111RRRR} \textcolor{green}{.RRRRRRRR.RRRRRRRR.RRRRRRRR}</td>
</tr>
</tbody>
</table>

\textcolor{red}{N} network ID bits \quad \textcolor{green}{M} multicast address bit
\textcolor{red}{H} host ID bits \quad \textcolor{green}{R} reserved bits
Asynchronous messaging using sockets

Internet protocol

IP v6 address

Addresses encoded on 128 bits

$2^{128} > 3.4 \times 10^{38}$ available addresses

$2.25 \times 10^{26}$ km

$\approx 10^{18} \times$ distance earth-sun

$\approx 10 \times$ earth-moon

$3.76 \times 10^6$ km

$10$ km
asynchronous messaging using sockets

addressing applications

an ip address designates a machine

a port number designates an application within a machine

this is known as port multiplexing

tcp or udp

application

process

port number

ip

machine

ip address

network

port number

ip address

machine

port number

ip address

machine

port number

ip address

...
asynchronous messaging using sockets

Sockets are programming abstractions representing bidirectional communication channels between processes.

There exists two types of sockets, namely tcp sockets and udp sockets.

In Java, sockets are instances of various classes found in the java.net package.
asynchronous messaging using sockets

transmission control protocol

stream

...bla bla bla bla bla... 

user datagram protocol

datagrams

tcp and udp exhibit dual features

<table>
<thead>
<tr>
<th></th>
<th>connection oriented</th>
<th>reliable channels</th>
<th>fifo ordering</th>
<th>message oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>UDP</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>
asynchronous messaging using sockets

tcp sockets

since tcp is connection-oriented, we have two classes for tcp sockets in java

client

```java
public class Socket {
    :
    public Socket(String host, int port) {...}
    public OutputStream getOutputStream() {...}
    public InputStream getInputStream() {{...}
    public void close() {...}
    :
}
```

server

```java
public class ServerSocket {
    :
    public ServerSocket(int port) {...}
    public Socket accept() {...}
    :
}
```

this captures the asymmetry when establishing a communication channel
public class DictionaryServer {
    private static Map dico = Map.of("inheritance", "héritage", "distributed", "réparti");

    public static void main(String[] args) {
        ServerSocket connectionServer = null;
        Socket clientSession = null;
        PrintWriter out = null;
        BufferedReader in = null;
        try {
            connectionServer = new ServerSocket(4444);
            clientSession = connectionServer.accept();
            out = new PrintWriter(clientSession.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(clientSession.getInputStream()));
            String word, mot;
            while ((word = in.readLine()) != null) {
                mot = (String) dico.get(word);
                if (mot == null) {
                    mot = "sorry, no translation available for " + word + "!";
                }
                out.println(mot);
            }
            out.close(); in.close(); connectionServer.close(); clientSession.close();
        } catch (IOException e) {
            System.out.println(e);
            System.exit(1);
        }
    }
}
public class DictionaryClient {
    public static void main(String[] args) {
        Socket mySession = null;
        PrintWriter out = null;
        BufferedReader in = null;
        BufferedReader stdIn = null;
        try {
            if (args.length < 1) {
                System.out.println("Hostname missing.");
                System.exit(1);
            }
            mySession = new Socket(args[0], 4444);
            out = new PrintWriter(mySession.getOutputStream(), true);
            in = new BufferedReader(new InputStreamReader(mySession.getInputStream()));
            stdIn = new BufferedReader(new InputStreamReader(System.in));
            String fromServer, fromUser;
            System.out.println("Go on, ask the dictionary server!");
            while (!((fromUser = stdIn.readLine()).equals("quit"))) {
                out.println(fromUser);
                fromServer = in.readLine();
                System.out.println("-> " + fromServer);
            }
            out.close(); in.close(); stdIn.close(); mySession.close();
        } catch (UnknownHostException e) {
            System.err.println("Host Unknown: " + args[0]);
            System.exit(1);
        } catch (IOException e) {
            System.err.println("No connection to: " + args[0]);
            System.exit(1);
        }
    }
}

**asynchronous messaging using sockets**

- tcp
- sockets
- client

📞

t

tcp

sockets

client
asynchronous messaging using sockets

the concept of streams (unix and java)

streams offer a unified programming abstraction for reading and writing data

streams can encapsulate various types of data sources, e.g., files, byte arrays in memory, sockets, etc.

streams can encapsulate other streams to stack up processing of the data

in java, streams are instances of various classes found in the java.io package
asynchronous messaging using sockets

the concept of streams (unix and java)

Socket clientSession = connectionServer.accept();
BufferedReader in = new BufferedReader(new InputStreamReader(clientSession.getInputStream()));

data source
byte stream
character stream
buffered character stream

printer and writer classes are special streams manipulating only characters

standard operating systems-level input and output streams are also accessed via java streams (System.in & System.out)
asynchronous messaging using sockets

the concept of object streams

fact the network knows nothing about objects, only bytes

problem so how can we send an object graph across the network?

solution any java object can be encoded into a stream of bytes and recreated from it by implementing the java.io.Serializable interface

this process is known as object serialization

```
ObjectOutputStream out = new ObjectOutputStream(sessionWithServer.getOutputStream());
out.writeObject(senderCollection);

ObjectInputStream in = new ObjectInputStream(sessionWithClient.getInputStream());
Collection receiverCollection = (Collection) in.readObject();
```
asynchronous messaging using sockets

since udp is connectionless, we have only one class for udp sockets in java

```
public class DatagramSocket {
    :
    public DatagramSocket() {...}
    public DatagramSocket(int port) {...}
    public void send(DatagramPacket packet) {...}
    public void receive(DatagramPacket packet) {...}
    public void close() {...}
    :
}
```

the **DatagramPacket** class is key when working with udp sockets

```
public class DatagramPacket {
    :
    public DatagramPacket(...) {...}
    public byte[] getData() {...}
    public InetAddress getAddress() {...}
    :
}
```

it captures the message-oriented nature of udp sockets
asynchronous messaging using sockets

public class QuoteServer {
    public static void main(String[] args) throws Exception {
        DatagramSocket socket = null;
        BufferedReader in = null;
        socket = new DatagramSocket(4445);
        in = new BufferedReader(new FileReader("one-liners.txt"));
        String quote = null;
        boolean moreQuotes = true;

        while (moreQuotes) {
            byte[] buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            quote = in.readLine();
            if (quote == null) {
                moreQuotes = false;
                buf = ("No more, bye!").getBytes();
            } else {
                buf = quote.getBytes();
                InetAddress address = packet.getAddress();
                int port = packet.getPort();
                packet = new DatagramPacket(buf, buf.length, address, port);
                socket.send(packet);
            }
        }
        socket.close();
    }
}
public class QuoteClient {
    public static void main(String[] args) throws Exception {
        if (args.length != 1) { System.out.println("Missing hostname"); System.exit(1); }
        DatagramSocket socket = new DatagramSocket();
        InetAddress address = InetAddress.getByName(args[0]);
        BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Go on, ask for a quote by typing return!");
        while (!stdIn.readLine().equals("quit")) {
            byte[] buf = new byte[256];
            DatagramPacket packet = new DatagramPacket(buf, buf.length, address, 4445);
            socket.send(packet);
            packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String received = new String(packet.getData()).trim();
            System.out.println("-> " + received);
        }
        socket.close();
    }
}
asynchronous messaging using sockets

one-to-one communication

one-to-many communication

udp multicast

a multicast address lies in range \([224.0.0.0, 239.255.255.255]\)
and defines a multicast group

in java, udp multicast is accessed via MulticastSocket, a subclass of DatagramSocket
messaging

asynchronous

using sockets

udp multicast

maximum number of routers a multicast packet can go through before being deleted

```java
public class MulticastQuoteSender {
    public static void main(String[] args) throws Exception {
        MulticastSocket socket = null;
        BufferedReader in = null;
        socket = new MulticastSocket();
        InetSocketAddress group = new InetSocketAddress(InetAddress.getByName("228.0.0.4"), 9000);
        NetworkInterface networkInterface = NetworkInterface.getByName("en0");
        socket.setTimeToLive(1);
        in = new BufferedReader(new FileReader("one-liners.txt"));
        String quote = null;
        boolean moreQuotes = true;

        while (moreQuotes) {
            Thread.currentThread().sleep(2000);
            byte[] buf = new byte[256];
            quote = in.readLine();
            if (quote == null) {
                moreQuotes = false;
                buf = ("No more, bye!").getBytes();
            } else {
                buf = quote.getBytes();
            }
            System.out.println("About to multicast: " + new String(buf));
            DatagramPacket packet = new DatagramPacket(buf, buf.length, group);
            socket.send(packet);
        }
        socket.close();
    }
}
```
asynchronous messaging using sockets

public class MulticastQuoteReceiver {

    public static void main(String[] args) throws Exception {
        try (MulticastSocket socket = new MulticastSocket(9000)) {
            InetSocketAddress group = new InetSocketAddress(InetAddress.getByName("228.0.0.4"), 9000);
            NetworkInterface netInterface = NetworkInterface.getByName("en0");
            socket.joinGroup(group, netInterface);
            while (true) {
                byte[] buf = new byte[256];
                DatagramPacket packet = new DatagramPacket(buf, buf.length);
                System.out.print("Waiting for the next quote: ");
                socket.receive(packet);
                String received = new String(packet.getData());
                System.out.println(received.trim());
                if (received.contains("bye")) {
                    break;
                }
            }
            socket.leaveGroup(group, netInterface);
            socket.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}