Tehnologii si Framework-uri Enterprise

Lecture 1
Outline

- Overview
- Concurrency
- Networking

Teachers:

- Lect. Ph.D. Radu Găceanu

Schedule:

- Lectures: 4 hours/week
- Lab: 4 hours/week

Contact:

- email
- group

Grading

- Lab
- Practical test
- Written exam
- Activity (bonus points)

Topics

- Networking
- IoC Containers
- Remoting
- ORM Frameworks
- Web services
- Web applications
- ...
- **Bibliography**: see *readme* file

Concurrency

examples

Sockets

`java.net`

TCP and UDP

- UDP (User Datagram Protocol): connectionless (additional data must be sent each time), datagram size limit - 64KB, unreliable (datagram receiving order is not guaranteed)
- TCP (Transmission Control Protocol): connection oriented, no limit, reliable

`ServerSocket` - represents a server socket that runs on the server and listens for incoming TCP connections
`Socket` - the class for performing client side TCP operations

Steps that occur when establishing a TCP connection between two computers using sockets:

1. The server instantiates a *ServerSocket* object, specifying which port number communication is to occur on.
2. The server invokes the *accept*() method of the *ServerSocket* class. This method waits until a client connects to the server on the given port.
3. While the server is waiting, a client instantiates a *Socket* object, specifying the server name and port number to connect to.
4. The constructor of the Socket class attempts to connect the client to the specified server and port number. If communication is established, the client now has a Socket object capable of communicating with the server.
5. On the server side, the accept() method returns a reference to a new *Socket* that is connected to the client's *Socket*.
6. Communication can now occur using I/O streams. Each socket has both an *OutputStream* and an *InputStream*. The client's *OutputStream* is connected to the server's *InputStream*, and the client's *InputStream* is connected to the server's *OutputStream*.

Server

```
ServerSocket server=null;
try{
    server=new ServerSocket(1234);
    while(keepProcessing){
        Socket client=server.accept(); //blocks and
    //waits for clients

        //processing code
    }
}catch(IOException ex){
    //...
}finally{
    if(server!=null){
        try{
            server.close();
        }catch(IOException ex){...}
    }
}
```

Client

```
try (Socket connection=new Socket("localhost", 1234)){
    //processing code
}catch(UnknownHostException e){
    //...
}catch(IOException e){
    //...
}
```

example

RPC (Remote Procedure Call)

- RPC is an inter-process communication paradigm that allows a computer program to cause a subroutine or procedure to execute in another address space without the programmer explicitly coding the details for this remote interaction
- The programmer writes almost the same code whether the subroutine is local to the executing program, or remote
- When the software is written using object-oriented paradigm, RPC may be referred to as remote invocation or Remote Method Invocation (**RMI**)
- RPC is a popular paradigm for implementing the client-server model of distributed computing
- RPC is initiated by a client sending a request message to a known remote server in order to execute a specified procedure using supplied parameters. A response is returned to the client and the application continues along with its process
- While the server is processing the call, the client is, by default, blocked (it waits until the server has finished processing before resuming execution)

example