# Programiranje Slozenih Softverskih Sistema

# SOCKETS CHEATSHEET

## Datagram - UDP - Blokirajuci

`SERVER`

```
 DatagramSocket s = new DatagramSocket(1234);
byte[] buffer = new byte[256];
DatagramPacket p = new DatagramPacket(buffer, buffer.lenght);
s.recive(p); // Ovde se blokira i ceka da primi paket
for(byte b: p.getData()) {}
```

`CLIENT`

```
 DagramSocket s = new DatagramSocket();
DatagramPacket p = new DatagramPacket(message.getBytes(), message.getBytes().length, INetAddress.getByName("localhost"), 1234);
s.send(p);
```

## StreamSocket - TCP - Blokirajuci

`SERVER`

```
 ServerSocket ss = new ServerSocket(1234);
Socket s = ss.accept(); // Ovde se blokira i ceka da primi paket
byte[] buffer = new byte[256];
s.getInputStream().read(buffer);
for(byte b: buffer) {}

// Ako koristimo niti, svaka nit ce raditi sa posebnim klijentom.
HashMap<String, BufferedWriter> clients = new HashMap<String, BufferedWriter>();
while(true) {
    Socket s = ss.accept();
    ServerThreadForClient nit = new ServerThreadForClient(clients, client);
    nit.run();
}
```

`CLIENT`

```
Socket s = new Socket(INetAddress.getByName("localhost"), 1234);
s.getOutputStream().write(message.getBytes());
s.close() // Veza postoji sve dok je neko ne ugasi

// Ako koristimo niti
BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));

Scanner scanner = new Scanner(System.in);
Thread toServer = new Thread(new Runnable() {

    @Override
    public void run() {
        String line = "";
        System.out.println("Unesite ime: ");
        while(true) {
            line = scanner.nextLine();
            try {
                bw.write(line + "\n");
                bw.flush();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
});
toServer.start();
```

`Thread`

```
public void run() {
    BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(client.getOutputStream()));
    BufferedReader br = new BufferedReader(new InputStreamReader(client.getInputStream()));

    name = br.readLine();

    synchronized (clients) {
        clients.put(name, bw);
    }

    while(true) {
        message = br.readLine();
        for (BufferedWriter bw_client : clients.values()) {
            bw_client.write(name + " : "+ message);
            bw_client.flush();
        }
    }
}
```

## MulticastSocket - UDP

IP address mora da bude tipa D od `224.0.0.0` do `239.255.255.255`

`Server`

```
InetAddress address = InetAddress.getByName(ADDRESS);
MulticastSocket ms = new MulticastSocket(PORT);
ms.setTimeToLive(2);
ms.joinGroup(address);

DatagramPacket p = new DatagramPacket(message.getBytes(), message.getBytes().length, address, PORT);
ms.send(p);
```

`Client`

```
InetAddress address = InetAddress.getByName(Server.ADDRESS);
MulticastSocket ms = new MulticastSocket(Server.PORT);
ms.joinGroup(address);

byte[] buffer = new byte[1024];
DatagramPacket p = new DatagramPacket(buffer, buffer.length);
ms.receive(p);
p.getData();
```

## RMI - Remote Method Invocation

Interfejs extends `Remote` i throws `RemoteException` - `IClassE` Class extends `UnicastRemoteObject` i implents interfejs - `ClassE`

`Server`

```
ClassE objectE = new ClassE();
LocateRegistry.createRegistry(Server.PORT);
Naming.rebind("//localhost" + ":" + Server.PORT + "/Server", objectE);
```

`Client`

```
IClassE server = (IClassE)Naming.lookup("rmi:" + "//localhost" + "/Server");
System.out.println(server.count());
```

## XML

`XML`

```

// Uzmi objekat
Socket s = new Socket(InetAddress.getByName("localhost"), Server.PORT);
BufferedWriter bw = new BufferedWriter(new OutputStreamWriter(s.getOutputStream()));
BufferedReader br = new BufferedReader(new InputStreamReader(s.getInputStream()));

String line = br.readLine();
XMLDecoder decoder = new XMLDecoder(new ByteArrayInputStream(line.getBytes()));
Object o = decoder.readObject();


// Posalji objekat
public String toString() {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    XMLEncoder encoder = new XMLEncoder(baos);
    encoder.writeObject(this);
    encoder.close();

    String s = new String(baos.toByteArray());
    s=s.replace("\n", " ");
    return s;
}

bw.write(classE.toString() + "\n");
```

# JSF CHEATSHEET

## DATABASE

`IProvider` - Interface

```
String DRIVER = "com.mysql.jdbc.Driver";
String CONNECTION_URL = "jdbc:mysql://localhost:3306/database?user=root&pass=";
```

`ConnectionProvider` - Class

```
private static Connection conn = null;
static {
    Class.forName(IProvider.DRIVER);
    conn = DriverManager.getConnection(IProvider.CONNECTION_URL);
}
public static Connection getConn() {
    return conn
}
```

`Statement sa bazom` - Class

```
Statement s = ConnectionProvider.getConn().createStatement();
ResultSet rs = s.executeQuery("SELECT * FROM user");
rs.next();
rs.getInt("id");
rs.getString("name");
```

`PreparedStatement sa bazom` - Class

```
PreparedStatement ps = ConnectionProvider.getConn().prepareStatement("SELECT * FROM user WHERE id=? AND name=?");
ps.setInt(1, id);
ps.setString(2, name);
ResultSet rs = ps.executeQuery();
---
ps.executeUpdate();
```

## JSF

`Beans` - Class

```
@ManagedBeans(name = "user")
@RequestScoped
---
obavezni getter-i i setter-i
```

`Sesija u klasi` - Class

```
private Map<String, Object> sessionMap = FacesContext.getCurrentContext().getExternalContext().getSessionMap();

sessionMap.put("new_user", this);
sessionMap.get("new_user");
sessionMap.remove("new_user");
```

`Citanje sesije i parametara iz URL-a u JSF` - xhtml

```
value=#{new_user} // iz sessije
value=#{param['id']} // iz parametara URL-a
```

`Validacija na JSF` - xhtml

```
xmlns:f="http://java.sun.com/jsf/core" // ubaciti
<f:validateRegex pattern="^[a-zA-Z]+(.)?[\s]*$" />
<f:validateRequired />
```

`Validacija na serveru` - Class

```
String pass = "Pass1";
pass.matches(".*[A-Z].*") // Sadrzi velika slova
pass.matches(".*[a-z].*") // Sadrzi mala slova
pass.matches(".*[0-9].*") // Sadrzi brojeve
```

`Error message` - Class

```
FacesContext.getCurrentInstance().addMessage("form:error", new FacesMessage("Poruka"));
```

`Error message` - xhtml

```
<h:message for="error" />
```

`Link ka drugoj stranici`

```
"index.xhtml?faces-redirect=true"
```

`IF statement u JSF`

```
<h:form rendered="#{uslov eq 'nesto'}">
</h:form>
eq ne gt and or
```

## JSF FORMA

```
<h:form id="form">
    <h:outputLabel for="name" />
    <h:inputText id="name" value="#{user.name}" validator="">
        <f:validateRequired />
    </h:inputText>

    <h:selectOneRadio value="#{user.gender}">
        <f:selectItem itemValue="Male" itemaLabel="Male" />
        <f:validateRequired />
    </h:selectOneRadio>

    <h:commandButton value="Save" action="#{user.save()}" />
    <h:link outcome="strana.xhtml?faces-redirect=true" value="Druga strana">
</h:form>
```

## JSF DATATABLE

```
<h:form id="form">
    <h:dataTable binding="#{table}" value="#{user.list()}" var="u">
        <h:column>
            <f:facet name="header">ID</f:facet>
            <h:outputText value="#{user.id}" />
        </h:column>
    </h:dataTable>
</h:form>
```

## UPLOAD

`JSF` - xhtml

```
<h:form id="form" enctype="multipart/form-data">
    <h:inputFile id="input_file" value="#{upload.file}">
        <f:ajax listener="#{upload.save()}" />
    </h:inputFile>
</h:form>
```

`Upload` - class

```
File file;
public void save() {
    try(InputStream input = file.getInputStream()) {
        Files.copy(input, new File("putanja sa \\", file.getSubmitedFilename)).toPath());
    } catch() {

    }
}
```