Servlet

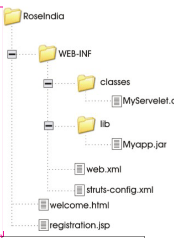
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
            http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
        version="4.0">
    <servlet>
        <servlet-name>EchoServlet</servlet-name>
        <servlet-class>ch.fhnw.ds.servlet.echo.EchoServlet</servlet-class>
        <init-param>
            <param-name>title</param-name>
            <param-value>Echo Servlet</param-value>
        </init-param>
    </servlet>
    <servlet-mapping>
        <servlet-name>EchoServlet</servlet-name>
        <url-pattern>/echo</url-pattern>
    </servlet-mapping>
</web-app>
```

```java
@WebServlet(name="EchoServlet", urlPatterns={„/echo/*“})
public class EchoServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException, ServletException {
        response.setContentType(„text/html“);
        PrintWriter out = response.getWriter();
        out.printf(„<html><body><b>%s</b></br><pre>%n“);
        out.println(„Properties:“);
        out.println(„getMethod:        „ + request.getMethod());
        out.println(„getContentLength: „ + request.getContentLength());
        out.println(„getContentType:   „ + request.getContentType());
        out.println(„getProtocol:      „ + request.getProtocol());
        out.println(„getRemoteAddr:    „ + request.getRemoteAddr());
        out.println(„getRemotePort:    „ + request.getRemotePort());
        out.println(„getRemoteHost:    „ + request.getRemoteHost());
        out.println(„getRemoteUser:    „ + request.getRemoteUser());
        out.println(„getServerName:    „ + request.getServerName());
        out.println(„getAuthType:      „ + request.getAuthType());
        out.println(„getQueryString:   „ + request.getQueryString());
        out.println(„getRequestURI:    „ + request.getRequestURI());
        out.println(„getRequestURL:    „ + request.getRequestURL());
        out.println(„getServletPath:   „ + request.getServletPath());
        out.println(„getContextPath:   „ + request.getContextPath());
```

JSON

```java
public class LocationClient {
    public static void main(String[] args) {
        Client c = ClientBuilder.newClient();
        WebTarget r = c.target(„http://transport.opendata.ch/v1/locations?x=47.482&y=8.211785“);
        String response = r.request().accept(MediaType.APPLICATION_JSON).get(String.class);
        Gson gson = new Gson();
        Locations locations = gson.fromJson(response, Locations.class);
        locations.getStations().stream().forEach((s) -> {
            System.out.printf(„Location: %s [%s/%s] Distance: %s -> https://maps.google.com/maps?q=loc:%s,%s\n“,
                    s.getName(), s.getCoordinate().getX(), s.getCoordinate().getY(), s.getDistance(),
                    s.getCoordinate().getX(), s.getCoordinate().getY());
        });
    }
}
```



GET	Mit GET wird die Repräsentation einer Ressource abgefragt. Da nur lesend zugegriffen wird, kann eine Ressource auf dem Server nicht verändert werden. Die HTTP-Methode GET unterstützt ein sehr effizientes Caching. **Expirationsmodell**: Vom Server gesendete HTTP-Header enthalten Informationen über die Gültigkeitsdauer einer Antwort. Innerhalb dieses Zeitraums kann der Client die zwischengespeicherte Antwort wiederver wenden, ohne mit dem Server erneut zu kommunizieren. **Validierungsmodell**: Der Client fragt beim Server an, ob eine bereits gelieferte Antwort wiederverwendet werden kann. Hierdurch wird dann ggf. eine Übertragung der Antwortdaten vermieden.
POST	Mit POST kann eine neue Ressource mit einem vom Server bestimmten URI erstellt werden. Mit POST kann aber auch eine beliebige Verarbeitung auf dem Server angestoßen werden.
PUT	PUT wird verwendet, um eine Ressource, deren URI bereits bekannt ist, zu erstellen oder zu ändern.
DELETE	Mit DELETE wird eine Ressource gelöscht.
HEAD	Im Unterschied zu GET wird mit HEAD keine Repräsentation zurückgeliefert, sondern nur Metadaten über die Ressource (Status-Code, Header). Hiermit kann z. B. die Existenz einer Ressource geprüft werden.
OPTIONS	OPTIONS liefert Informationen über eine Ressource, z. B. welche Repräsentationsformate unterstützt werden und welche Methoden erlaubt sind.
PUT vs PATCH	PUT ist eine idempotente Operation. PATCH ist nicht zwingend idempotent. Eine Entität welche mit PATCH umschlossen ist, enthält ein Set von Instruktionen, welche beschreiben wie die Ressource verändert wird.

REDIRECT

```java
public class MySearcher extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletRespon-
se response)
            throws ServletException, IOException {
        String name=request.getParameter(„name“);
        response.sendRedirect(„https://www.google.co.in/#q=“+name);
    }
}
```

```java
URL url = new URL(„http://localhost:80/bank“);
HttpURLConnection c = (HttpURLConnection)url.openConnection();
c.setRequestMethod(„POST“);
c.setDoOutput(true);
c.setDoInput(true);
c.connect();
OutputStream os = c.getOutputStream();
OutputStreamWriter wr = new OutputStreamWriter(os);
wr.write(„user=Dominik2&amount=1000“); wr.flush();
InputStream is = c.getInputStream();
BufferedReader rd = new BufferedReader(new InputStreamReader(is));
rd.lines().forEach(System.out::println);
```

```
– 100 Continue
– 101 Switching Protocols
– 200: OK
– 201: Created
– 202: Accepted
– 204: No Content
– 300: Multiple Choices
– 301: Moved Permanently
– 302: Found
– 303: See Other (e.g. after
POST)
– 304: Not Modified
– 307: Temporary Redirect
– 400: Bad Request
– 401: Unauthorized
– 403: Forbidden
– 404: Not Found
– 405: Method Not Allowed
– 406: Not Acceptable
– 408: Request Time-out
– 411: Length Required
– 413: Request Entity Too Large
– 414: Request-URI Too Large
– 415: Unsupported Media Type
– 500: Internal Server Error
– 501: Not Implemented
```

REST

```java
@Singleton
@Path(„/msg“)
public class MsgResource {

    private String msg = „Hello, world!“;
    public MsgResource() {
        System.out.println(„MsgResource() called“);
    }
    @GET
    @Produces(„text/plain“)
    public String getPlain() {
        return msg + „\n“;
    }
    @GET
    @Produces(„text/html“)
    public String getHtml() {
        StringBuffer buf = new StringBuffer();
        buf.append(„<html><body><h1>Message Text</h1>“ + msg + „<br>“);
        buf.append(„<form method=\“POST\“ action=\“/msg\“>“);
        buf.append(„<p>Text: <input name=\“msg\“ type=\“text\“ size=20/>“);
        buf.append(„<input type=\“submit\“ value=\“Submit\“ />“);
        buf.append(„</form>“);
        buf.append(„</body></html>“);
        return buf.toString();
    }
    @GET
    @Produces(„text/xml“)
    public String getSimpleXml() {
        return „<string>“ + msg + „</string>“;
    }
    @GET
    @Produces( { „application/json“, „application/xml“, „application/xstream“ })
    public Msg getXml() {
        return new Msg(msg);
    }
    @PUT
    @Consumes(„text/plain“)
    public void setTextPlain(String new_msg) {
        msg = new_msg;
    }
    @PUT
    @Consumes( { „application/json“, „application/xml“, „application/xstream“ })
    public void setTextXml(Msg message) {
        msg = message.getText();
    }
    @POST
    @Consumes(„application/x-www-form-urlencoded“)
    @Produces(„text/html“)
    public String doPost(@FormParam(„msg“) String new_msg) {
        msg = new_msg;
        return getHtml();
    }
    private final AtomicInteger counter = new AtomicInteger();
    private final Map<Integer, Msg> map = new HashMap<>();
    @POST
    @Consumes({ „application/xml“, „application/json“, „application/xstream“ })
    public Response createNewMessage(@Context UriInfo uriInfo, Msg message) {
        int id = counter.getAndIncrement();
        map.put(id, message);
        URI location = uriInfo.getAbsolutePathBuilder().path(„“ + id).build();
        return Response.created(location).build();
    }
    @GET
    @Produces(„text/plain“)
    @Path(„{id}“)
    public String getData(@PathParam(„id“) int id) {
        return map.get(id).getText() + „\n“;
    }
}
```

```java
public class Server {
    public static void main(String[] args) throws IOException {
        final String baseUri = „http://localhost:9998/“;
        final ResourceConfig rc = new ResourceConfig().packages(„ch.fhnw.ds.rest.msg.
resources“);
        System.out.println(„Starting grizzly...“);
        HttpServer httpServer = GrizzlyHttpServerFactory.createHttpServer(URI.create(ba-
seUri), rc);
        // WADL available at ${baseUri}/application.wadl
        System.out.println(„Hit enter to stop it...“);
        System.in.read();
        httpServer.shutdown();
    }
}
```

```java
    @GET
    @Produces(„text/plain“)
    public String getAccountNumbers(@Context UriInfo uriInfo) throws IOException {
        StringBuffer buf = new StringBuffer();
        for (String acc : bank.getAccountNumbers()) {
            buf.append(uriInfo.getAbsolutePathBuilder().path(acc).build());
            buf.append(„\n“);
        }
        return buf.toString();
    }
```

IO-STREAM

```java
public class Write {
   public static void main(String[] args) throws Exception {
      ObjectOutputStream out = new ObjectOutputStream(
                                  new FileOutputStream(„data.txt“));
      Deposit cmd = new Deposit(4456, 130.50);
      out.writeObject(cmd);
      out.close();
   }
}

public class Read {
   public static void main(String[] args) throws Exception {
      try(ObjectInputStream in = new ObjectInputStream(new FileInputStream(„data.txt“))) {
            Object obj = in.readObject();
            if (obj instanceof Deposit) {
                System.out.println(„Deposit Object read:“);
                System.out.println(((Deposit) obj).getAccount());
                System.out.println(((Deposit) obj).getAmount());
            }
        }
    }
}
```

WEBSOCKET 1

```java
@ServerEndpoint(„/echo“)
public class Echo {
    private Session session;
    private int counter;
    @OnOpen
    public void open(Session session) {
        System.out.println(session.getId() + „: onOpen“);
        this.session = session;
        try {
            session.getBasicRemote().sendText(„Verbindung wurde hergestellt.“);
        } catch (IOException e) {}
    }
    @OnClose
    public void close() {
        System.out.println(session.getId() + „: onClose“);
    }
    @OnError
    public void error(Throwable error) {
        System.out.println(session.getId() + „: „ + error);
    }
    @OnMessage
    public void message(String msg) {
        try {
            if (session.isOpen()) {
                System.out.println(msg);
                session.getBasicRemote().sendText(++counter + „. „ + msg);
            }
        } catch (IOException e) {
            try {
                session.close();
            } catch (IOException e1) {}
        }
    }
}
```

REST

```java
@PUT
@Path(„{id}“)
@Produces({ „application/json“ })
@Consumes({ „application/json“ })
public Response putAccount(@Context Request request, @PathParam(„id“) String number, AccountDto dto)
        throws IOException {
    Account a = bank.getAccount(number);
    if (a == null)
        throw new NotFoundException(„Account not found“); // 404
    ResponseBuilder builder = request.evaluatePreconditions(new EntityTag(„“ + a.hashCode()));
    if (builder != null) {
        return builder.build();
    }
    double delta = dto.getBalance() - a.getBalance();
    if (delta != 0) {
        try {
            if (delta > 0)
                a.deposit(delta);
            else
                a.withdraw(-delta);
        } catch (IllegalArgumentException e) {
            throw new WebApplicationException(e, Response.Status.BAD_REQUEST);
        } catch (InactiveException e) {
            throw new WebApplicationException(e, Response.Status.GONE);
        } catch (OverdrawException e) {
            throw new WebApplicationException(e, Response.Status.FORBIDDEN);
        }
    }
    return Response.ok(new AccountDto(a)).tag(„“ + a.hashCode()).build();
}
```

Host	Domain name of the server (for virtual hosting) and the port number Mandatory since HTTP/1.1
Accept	Data types supported by the client (may be considered by the server or not)
Accept-Language	Human languages supported by the client
Accept-Encoding	Encodings supported by the client, e.g. gzip or deflate
User-Agent	browser details, supplies server with information about the type of browser making the request
Referer	URL from which the request is initiated. By checking the referrer, the new server can see where the request originated.
Connection	Type of connection the client would prefer (e.g. keep-alive or close); Example: keep-alive (browser requests use of persistent TCP connections)
Cookie	A cookie previously sent by the server
Content-Length	The length of the request body in bytes
Content-Type	The MIME type of the body of the request (for POST and PUT requests)
Content-Encoding	compression algorithms
Location	used by redirections
Date	modification date of resource (assumed by server)
Last-Modified	timestamp when the response was created
Expires	date after which the result is considered stale
Server	information about the server
Transfer-Encoding	specifies type of transformation (e.g. chunked)
Cache-Control	information about cache handling (e.g. no-cache disables caching)
WWW-Authenticate	information about authentication method

```
public class BankImpl extends java.rmi.server.UnicastRemoteObject implements bank.rmi.RmiBank {
    private final Bank bank;
    public BankImpl(Bank bank) throws RemoteException {
        this.bank = bank;
    }
    public String createAccount(String owner) throws IOException {
        return bank.createAccount(owner);
    }
    public boolean removeAccount(String number) throws IOException {
        return bank.removeAccount(number);
    }
    public Set<String> getAccountNumbers() throws IOException { // makes result set definitively serializable
        return new HashSet<String>(bank.getAccountNumbers());
    }
    public bank.Account getAccount(String number) throws IOException {
        bank.Account a = bank.getAccount(number);
        return a == null ? null : new AccountImpl(a);
        // AccountImpl is the exported RMI implementation
    }
    public void transfer(bank.Account a, bank.Account b, double amount)
            throws IOException, InactiveException, OverdrawException {
        bank.transfer(a, b, amount);
    }
}
```

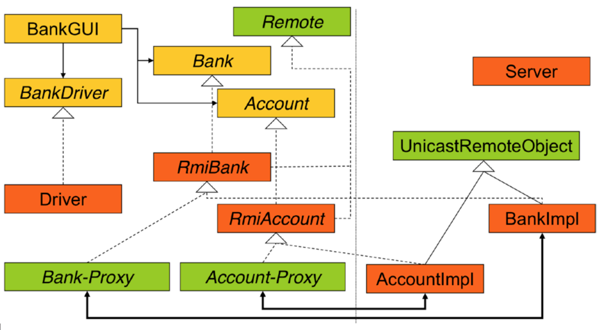
```
public class AccountImpl extends java.rmi.server.UnicastRemoteObject implements bank.rmi.RmiAccount {
private final Account acc;
public AccountImpl(Account a) throws RemoteException {
      this.acc = a;
   }
   public String getNumber() throws IOException { return acc.getNumber();}
   public String getOwner() throws IOException{return acc.getOwner();}
   public void deposit(double amount) throws IOException, InactiveException {acc.deposit(amount);}
}
```

```
class Server {
   public static void main(String[] args) throws Exception {
    // use e.g. local (threadsafe) bank
    Bank localBank = new bank.local.LocalBank(); // create and export RMI bank
    Bank bank = new BankImpl(localBank);
    Naming.rebind(„Bank“, bank); System.out.println(„Server started...“);
   }
}
```

```
public class Driver implements bank.BankDriver {
   private bank.Bank bank;
   public void connect(String[] args) throws IOException {
      String host = args[0];
      try {
         bank = (bank.Bank)Naming.lookup(„rmi://“ + host + „/Bank“);
      } catch(NotBoundException e) { throw new IOException(e);}
   }
   public void disconnect() {bank = null;}
   public bank.Bank getBank() {return bank;}
}
```

```
public class BankImpl extends UnicastRemoteObject
                      implements RmiBank {
   private final Bank bank;
   public BankImpl(Bank bank) throws RemoteException {
      this.bank = bank;
   }
   public String createAccount(String owner) throws IOException {
      String id = bank.createAccount(owner);
      if(id != null) notifyListeners(id);
      return id;
   }
   public boolean closeAccount(String number) throws IOException {
      boolean res = bank.closeAccount(number);
      if(res) notifyListeners(number);
      return res;
   }
}
```

```
public class AccountProxy implements Serializable, bank.Account {
   private final bank.rmi.RmiAccount impl;
   private final String number;
   private final String owner;
   AccountProxy(bank.rmi.RmiAccount impl) throws IOException {
     this.impl = impl;
     number = impl.getNumber();
     owner = impl.getOwner();
   }
   public String getNumber() { return number; }
   public String getOwner()  { return owner; }
   public boolean isActive() throws IOException {return impl.isActive();}
}
```



```
import java.net.*;
public class DSender{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket();
    String str = „Welcome java“;
    InetAddress ip = InetAddress.getByName(„127.0.0.1“);

    DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
    ds.send(dp);
    ds.close();
  }
}

import java.net.*;
public class DReceiver{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```

```
public class UploadClient {
 public static void main(String[] args) {
  String host = args[0];
  int port = Integer.parseInt(args[1]);
  String file = args[2];
  try (Socket socket = new Socket(host, port); BufferedReader in = new BufferedReader(new InputStream-
Reader(
   socket.getInputStream())); OutputStream out = socket.getOutputStream()) {
   try (InputStream is = new FileInputStream(file)) {
    byte[] buffer = new byte[8192];
    int c;
    while ((c = is.read(buffer)) != -1) {
     out.write(buffer, 0, c);
    }
   }
   socket.shutdownOutput();
   String msg = in .readLine();
   System.out.println(msg);
  } catch (Exception e) {
   System.err.println(e);
  }
 }
}
```

```
public class EchoServer4 {
 public static void main(String args[]) throws IOException {
  ServerSocket server = new ServerSocket(1234);
  ExecutorService pool = Executors.newCachedThreadPool();
  while (true) {
   Socket s = server.accept();
   pool.execute(new Task(s));
  }
 }
}
class Task implements Runnable {
 private Socket s;
 EchoHandler(Socket s) {
  this.s = s;
 }
 public void run() {
  try (BufferedReader in = new BufferedReader(new InputStreamReader(s.getInputS-
tream()));
   PrintWriter out = new PrintWriter(s.getOutputStream(), true)) {
   String input = in .readLine();
   while (input != null && !““.equals(input)) {
    out.println(input);
    input = in .readLine();
   }
   System.out.println(„done serving „ + s);
   s.close();
  } catch (IOException e) {
   System.err.println(e);
  }
 }
}
```

```
public class MarshalTest {
    public static void marshal(Object obj, String filename)
    throws JAXBException {
        JAXBContext context = JAXBContext.newInstance(obj.getClass());
        Marshaller marshaller = context.createMarshaller();
        marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
        File xmlFile = new File(filename);
        marshaller.marshal(obj, xmlFile);
    }
    public static void main(String[] args) throws JAXBException {
        Message message1 = new Message(1, „2015-01-15 13:41:05“, „Nachricht A“);
        Message message2 = new Message(2, „2015-01-15 14:00:00“, „Nachricht B“);
        List < Message > list = new ArrayList < Message > ();
        list.add(message1);
        list.add(message2);
        Messages messages = new Messages(list);
        System.out.println(messages);
        marshal(messages, „messages.xml“);
        marshal(message1, „message.xml“);
    }
}
public class UnmarshalTest {
    public static void main(String[] args) throws JAXBException {
        File xmlFile1 = new File(„messages.xml“);
        Messages messages = JAXB.unmarshal(xmlFile1, Messages.class);
        System.out.println(messages);
        File xmlFile2 = new File(„message.xml“);
        Message message = JAXB.unmarshal(xmlFile2, Message.class);
        System.out.println(message);
    }
}
```

```
@ClientEndpoint
public class EchoClient {
    private static final String URI = „ws://localhost:8080/webso-
cket/echo“;
    private Session session;
    @OnOpen
    public void open(Session session) {
        System.out.println(„onOpen“);
        this.session = session;
    }
    @OnClose
    public void close() {
        System.out.println(„onClose“);
    }
    @OnMessage
    public void message(String message) {
        System.out.println(message);
    }
    public void sendMessage(String message) throws IOException {
        if (session.isOpen())
            session.getBasicRemote().sendText(message);
    }
    public static void main(String[] args) {
        ClientManager clientManager = ClientManager.createClient();
        EchoClient client = new EchoClient();
        try {
            Session session = clientManager.connectToServer(client,
new URI(URI));
            for (int i = 0; i < 10; i++) {
                client.sendMessage(String.valueOf(Math.random()));
                Thread.sleep(1000);
            }
            session.close();
            clientManager.shutdown();
        } catch (DeploymentException | IOException | URISyntaxEx-
ception |
            InterruptedException e) {
            System.out.println(e.getMessage());
        }
    }
}
```