Software architecture

Week 12 - Web services (JAX-WS & JAX-RS)

Requirements

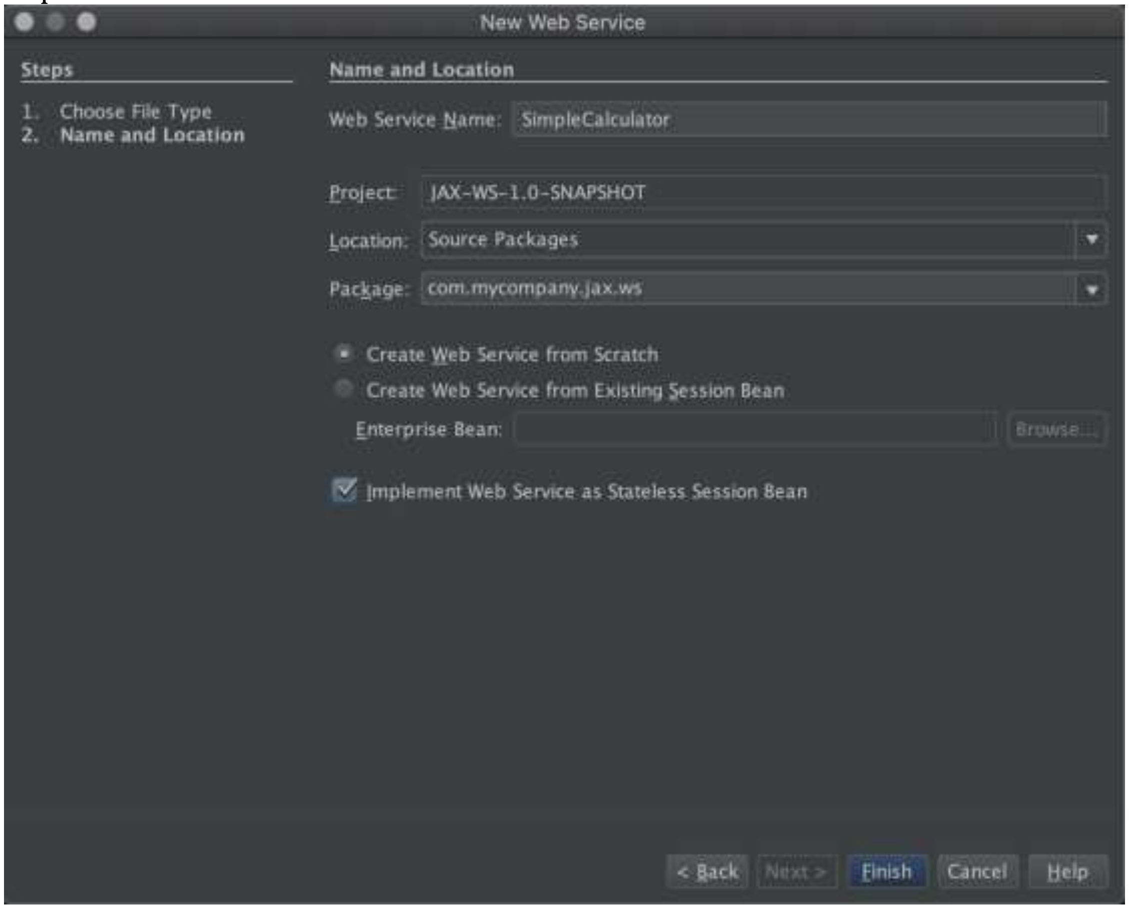
1. Netbeans 11
2. Java Development Kit 8
3. Payara Server

Webservices with JAX-WS

In this exercise, we will create a project that implements webservices using Java API for XML Web Services. In this project, we will deploy the Web Service and create a client that will submit a request to our web services and display the result.

Let's create a new project (Web Application) called `JAX-WS` in Netbeans.

To create a new Webservice, right-click on your project > New > Web Services > Web Service. Let's call it `SimpleCalculator` and let's save it under `com.mycompany.jax.ws` and implement the `Web Service as Stateless Session Bean`.



Web Service Name: SimpleCalculator

Project: JAX-WS-1.0-SNAPSHOT

Location: Source Packages

Package: com.mycompany.jax.ws

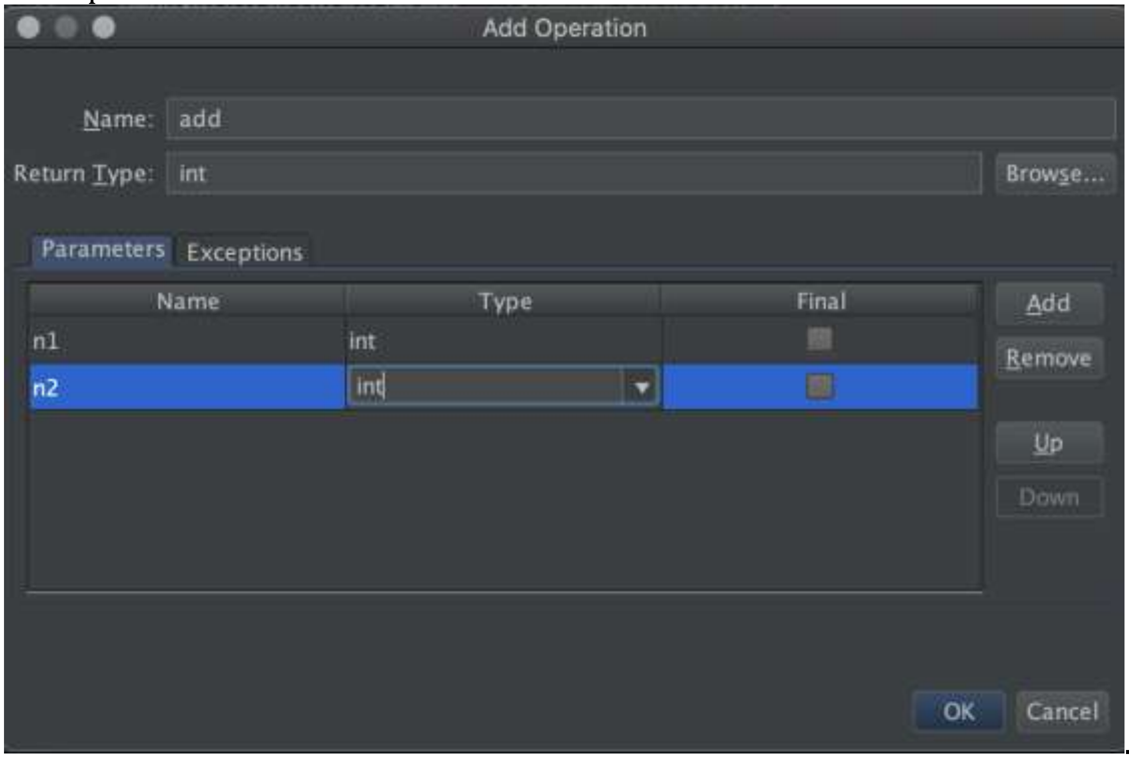
(●) Create Web Service from Scratch

( ) Create Web Service from Existing Session Bean

Enterprise Bean:

[✓] Implement Web Service as Stateless Session Bean

Add Operation...



Name: add

Return Type: int

Name	Type	Final
n1	int	☐
n2	int	☐

.

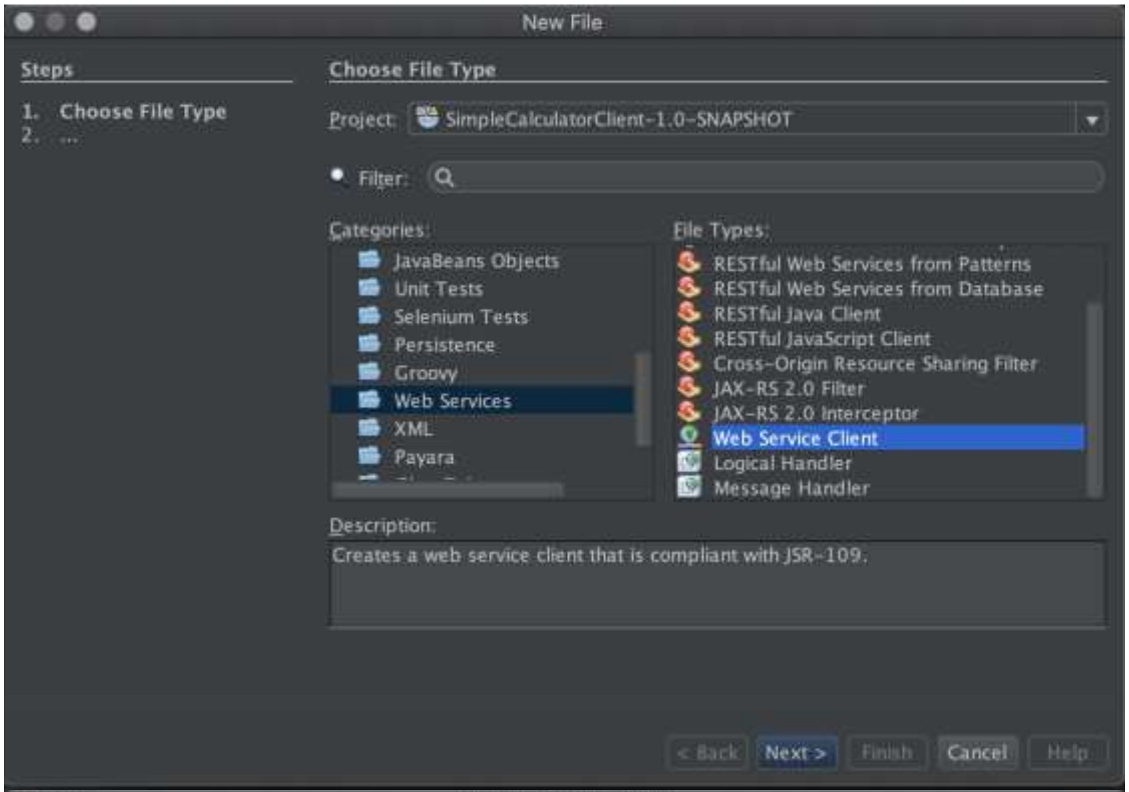
As we mentionned above, our newly generated WebMethods will take two inputs (n1 and n2).
Write the operation in the newly generated WebMethods and return the results.

To test your web service, start Payara Server (Services > Servers > Payara Server), right click on `SimpleCalculator` and click on `Test Web Service`.

Client application

Once the webservice is deployed, we can create a client that will make a request to the Web service and return the results of the operations.
We can either create a new Java Application or a Web Application. For this exercise session, we will go for a Web Application.

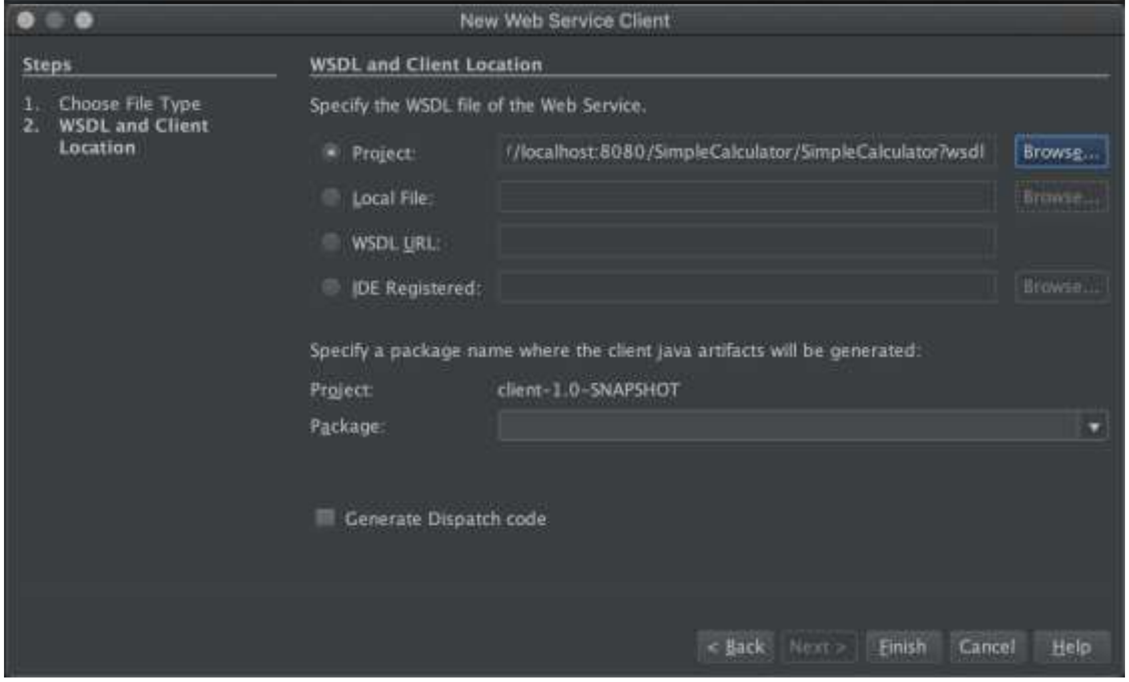
Create a new Web project called `SimpleCalculatorClient`.
To create a new WebService client, Right-click on your project > New > Other > Web Services > Web Service Client.
In the `Project`'s input field, select your project, then click on `Finish`.



Project: SimpleCalculatorClient-1.0-SNAPSHOT

Filter:

Description:
Creates a web service client that is compliant with JSR-109.



Specify the WSDL file of the Web Service.

Project: ʳ/localhost:8080/SimpleCalculator/SimpleCalculator?wsdl
Local File:
WSDL URL:
IDE Registered:

Specify a package name where the client java artifacts will be generated:

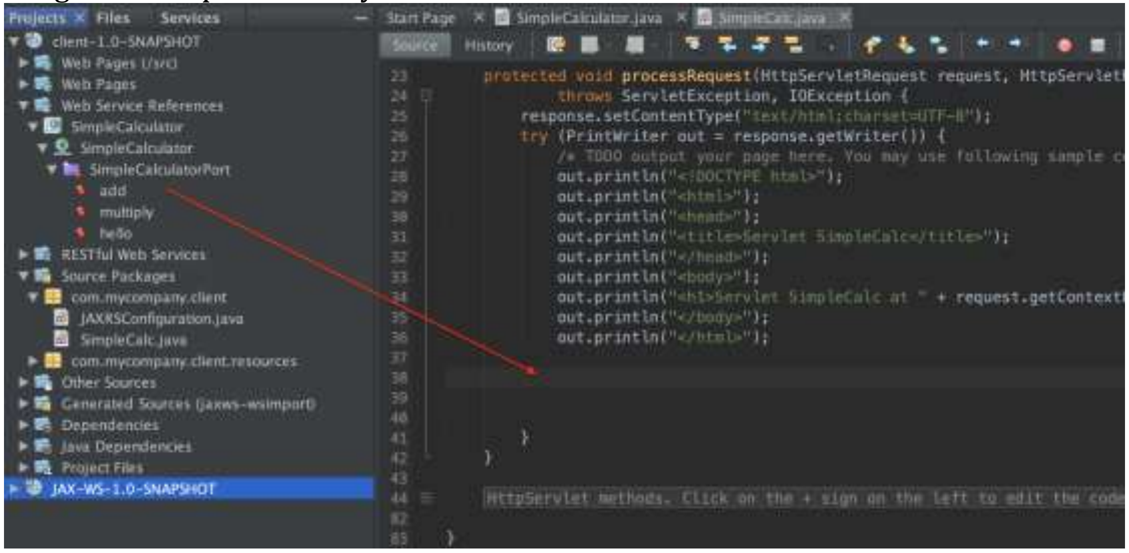
Project: client-1.0-SNAPSHOT
Package:

Generate Dispatch code

Now, you have to create a new Servlet to make a request to the Web Service. You can name it `ClientServlet`.

In the Web Service References node, expand the node that represents the web service. The `add` operation, which you will invoke from the client, is now exposed.

Drag the add operation to your servlet.



```
    protected void processRequest(HttpServletRequest request, HttpServlet
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        try (PrintWriter out = response.getWriter()) {
            /* TODO output your page here. You may use following sample c
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet SimpleCalc</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Servlet SimpleCalc at " + request.getContext
            out.println("</body>");
            out.println("</html>");



        }
    }

    HttpServlet methods. Click on the + sign on the left to edit the code

}
```

Run your project!

Common errors

- `package javax.jws does not exist`: Make sure that you use `jdk 1.8`. To change the JDK version for your project, you have to right-click on the project > Properties > Build > Compile > Java Platform. Select the right JDK version
- Package `com.mycompany.jax.ws...` does not exist: Clear the Netbeans cache.

Webservices with JAX-RS

1. Create a new web application, call it **JAX_RS**
2. If you still have BANKDB that we used in 6th week in JPA_BankExercise, connect to the BANKDB. If you already have it, you can skip to the 4th step. Otherwise, you should create a new database connection.
3. The database connection should have 2 tables, which are called **ACCOUNT** and **PAYMENT**
    1. **ACCOUNT** should have 4 fields:
        1. **ID** Type: REAL Constraints: Primary Key, Unique, and Index
        2. **LASTNAME** Type: VARCHAR Size: 255 Constraints: Null
        3. **FIRSTNAME** Type: VARCHAR Size: 255 Constraints: Null
        4. **BALANCE** Type: DOUBLE Default: 0.0 Constraints: Null

2. **PAYMENT** should have 3 fields:

    1. **ID** Type: REAL Constraints: Primary Key, Unique, and Index
    2. **EXECDATE** Type: DATE Constraints: Null
    3. **AMOUNT** Type: DOUBLE Default: 0.0 Constraints: Null

4. Right-click on the project >> New >> Other >> Web Services >> RESTful Web Services from Database
5. Choose your database connection, then choose both Account and Payment tables.
6. Write **ch.unil.doplab** as the package name and finish it.
7. Create 2 packages called **ch.unil.doplab.ejb** and **ch.unil.doplab.web**.
8. Inside the **ch.unil.doplab.ejb** package, create an interface and call it **BankBeanLocal**. Add 6 method signatures described below:

    1. **openAccount** : takes 2 `String` parameters (*lastName* and *firstName*), has `Account` return type.
    2. **closeAccount**: takes 1 `long` parameter (*accountID*), doesn't have any return type.
    3. **deposit**: takes 1 `long` parameter (*accountID*) and 1 `double` parameter (*amount*), has `double` return type.
    4. **withdraw**: takes 1 `long` parameter (*accountID*) and 1 `double` parameter (*amount*), has `double` return type.
    5. **findAccountByID**: takes 1 `long` parameter (*accountID*), has `Account` return type.
    6. **findAccountsByLastName**: takes 1 `String` parameter (*lastName*), has `List<Account>` return type.
    7. **findAllAccounts**: doesn't take any parameters, has `List<Account>` return type.

9. Inside the **ch.unil.doplab.ejb** package, create a `@Stateless` EJB called **BankBean**, which implements the interface **BankBeanLocal**, described in the step 8.

    1. Then, add the following class annotation:

    ```
    @TransactionManagement(javax.ejb.TransactionManagementType.CONTAINE
    R)
    ```

    2. Add 2 instance variables:

    ```
    @Resource
    SessionContext context;
    ```

```
@PersistenceContext
private EntityManager manager;
```

3. Override the 7 methods which come from the interface, **BankBeanLocal**.

```
@Override
public Account openAccount(String lastName, String firstName) {
  Account account = new Account(lastName, firstName);
  System.out.println("before persisting ->" + account);
  manager.persist(account);
  System.out.println("after persisting ->" + account);
  return account;
 }

@Override
public void closeAccount(long accountID) {
  Account account = manager.find(Account.class, accountID);
  manager.remove(account);
}

@Override
public double deposit(long accountID, double amount) {
  Account account = manager.find(Account.class, accountID);
  return account.deposit(amount);
}

@Override
public double withdraw(long accountID, double amount) {
  Account account = manager.find(Account.class, accountID);
  if (account.getBalance() - amount < 0) {
    context.setRollbackOnly();
    throw new IllegalArgumentException("insufficient balance");
  }
  return account.withdraw(amount);
}

@Override
public Account findAccountByID(long accountID) {
  return manager.find(Account.class, accountID);
}

@Override
public List<Account> findAllAccounts() {
  Query query = manager.createQuery("SELECT a FROM Account a");
  return query.getResultList();
}

@Override
public List<Account> findAccountsByLastName(String lastName) {
    Query query = manager.createNamedQuery("findByLastName");
    return query.setParameter("lastName", lastName).getResultList();
  }
```

10. Inside the **ch.unil.doplab.ejb** package, create a `@Stateful` EJB called **AccountBean**.

    1. Add 2 instance variables:

    ```
    @PersistenceContext(type = PersistenceContextType.EXTENDED)
    private EntityManager manager;

    private Account account = null;
    ```

    2. Add 4 methods:

    ```
    public void open(int accountID) {
      account = manager.find(Account.class, accountID);
      if (account == null) {
        account = new Account();
        manager.persist(account);
      }
    }

    public void associate(int accountID) {}

    public void deposit(int amount) {
      if (account == null) throw new IllegalStateException();
      account.deposit(amount);
    }

    public String getLastName() {
      if (account == null) throw new IllegalStateException();
      return account.getLastName();
    }
    ```

11. Inside the **ch.unil.doplab.web** package, create a servlet called **BankServlet** and change its *processRequest(...)* as follows:

```
response.setContentType("text/html;charset=UTF-8");
    try (PrintWriter out = response.getWriter()) {
        /* TODO output your page here. You may use following sample code.
*/
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Banking – Output</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Servlet BankServlet at " +
request.getContextPath() + "</h1>");
```

```
        String operation = request.getParameter("action");
        out.println("---------------<br>");
        out.println("operation  = " + operation + "<br>");
        out.println("---------------<br>");
        if (operation.equals("open")) {
            String lastName = request.getParameter("lastname");
            String firstName = request.getParameter("firstname");
            out.println("opening an account for " + firstName + " " +
lastName + "<br>");
            Account account = theBank.openAccount(lastName, firstName);
            out.println("account =  " + account + "<br>");
            HttpSession mySession = request.getSession();
            if (mySession.isNew()) {
                mySession.setAttribute("lastName", lastName);
                mySession.setAttribute("firstName", firstName);
            } else {
                lastName = (String) mySession.getAttribute("lastName");
                firstName = (String) mySession.getAttribute("firstName");
            }
        } else if (operation.equals("close")) {
            long accountID =
Long.parseLong(request.getParameter("account"));
            Account account = theBank.findAccountByID(accountID);
            if (account == null) {
                out.println("account " + accountID + " does not
exist<br>");
            } else {
                theBank.closeAccount(accountID);
                out.println("closing account " + accountID + " of " +
account.getFirstName() + " " + account.getLastName() + "<br>");
            }
        } else if (operation.equals("deposit")) {
            long accountID =
Long.parseLong(request.getParameter("account"));
            double amount =
Double.parseDouble(request.getParameter("amount"));
            double balance = theBank.deposit(accountID, amount);
            Account account = theBank.findAccountByID(accountID);
            out.println("the new balance of " + account + " is $" +
balance + "<br>");
        } else if (operation.equals("withdraw")) {
            long accountID =
Long.parseLong(request.getParameter("account"));
            double amount =
Double.parseDouble(request.getParameter("amount"));
            double balance = theBank.withdraw(accountID, amount);
            Account account = theBank.findAccountByID(accountID);
            out.println("the new balance of " + account + " is $" +
balance + "<br>");
        } else if (operation.equals("find all accounts")) {
            List<Account> accounts = theBank.findAllAccounts();
            if (accounts.isEmpty()) {
                out.println("-> no accounts found <br>");
            } else {
                for (Account account : accounts) {
                    out.println("-> " + account + "<br>");
                }
            }
        } else if (operation.equals("find accounts by name")) {
            String lastName = request.getParameter("lastname");
            out.println("-> last name = " + lastName + "<br>");
            List<Account> accounts = 
theBank.findAccountsByLastName(lastName);
            if (accounts.isEmpty()) {
                out.println("-> no accounts found <br>");
            } else {
                for (Account account : accounts) {
                    out.println("-> " + account + "<br>");
                }
            }
        } else {
            out.println("unknown action<br>");
        }
        out.println("---------------<br>");
        out.println("</body>");
        out.println("</html>");
```

12. As for your **index.html** file, you can take the following code as it is:

```
<!DOCTYPE html>
<html>
<head>
    <title>Online Banking </title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
</head>
<body>
  Open Account<br><br>
  <form method="post" action="bank">
   Last Name   <input type="text" name="lastname" value="Simpson"><br>
   First Name  <input type="text" name="firstname" value="Marge"> <br>
     <input type="submit" name="action" value="open">
  </form>
  ------------------------------------------------------------<br>
  Close Account<br><br>
  <form method="post" action="bank">
   Account   <input type="text" name="account" value="1"><br>
     <input type="submit" name="action" value="close">
  </form>
  ------------------------------------------------------------<br>
  Transfer Money<br><br>
  <form method="post" action="bank">
   Account   <input type="text" name="account" value="1"><br>
   Amount  <input type="text" name="amount" value="100"> <br>
     <input type="submit" name="action" value="deposit">
     <input type="submit" name="action" value="withdraw">
  </form>
  ------------------------------------------------------------<br>
  List Accounts<br><br>
  <form method="post" action="bank">
   Last Name   <input type="text" name="lastname" value="Simpson"><br>
     <input type="submit" name="action" value="find all accounts">
     <input type="submit" name="action" value="find accounts by name">
  </form>

</body>
</html>
```

13. Run your code!

Exercises

- Using JAX-RS, write a program that takes a message as input and returns the encoded version using Cesar Cipher (Week 5)