Java EE: Beyond The Basics HOL1944

- David Delabassee
  Oracle
  @delabassee
- David R. Heffelfinger
  Ensode Technology, LLC
  http://www.ensode.com
  @ensode
- Bob Larsen
  Metaformers, Inc.
  @direHerring
- Sven Reimers
  Airbus Defence and Space
  @SvenNB

Session Outline

- JavaServer Faces (JSF)
  - HTML5 Markup
  - Facelets Templating
  - Faces Flows
  - Resource Library Contracts

Session Outline (con'td)

- CDI
  - Qualifiers
  - Stereotypes
  - Interceptor Binding Types
  - CDI Events

Session Outline (con'td)

- Enterprise JavaBeans (EJB's)
  - Interceptors
  - Lifecycle Annotations
  - Transaction Management
  - Asynchronous Method Invocation
  - EJB Timer Service

Session Outline (cont'd)

- Java API for RESTful Web Services (JAX-RS)
  - JAX-RS Overview
  - Generating JAX-RS Web Services from a Database
  - Generating RESTful web service clients

Hands On Labs

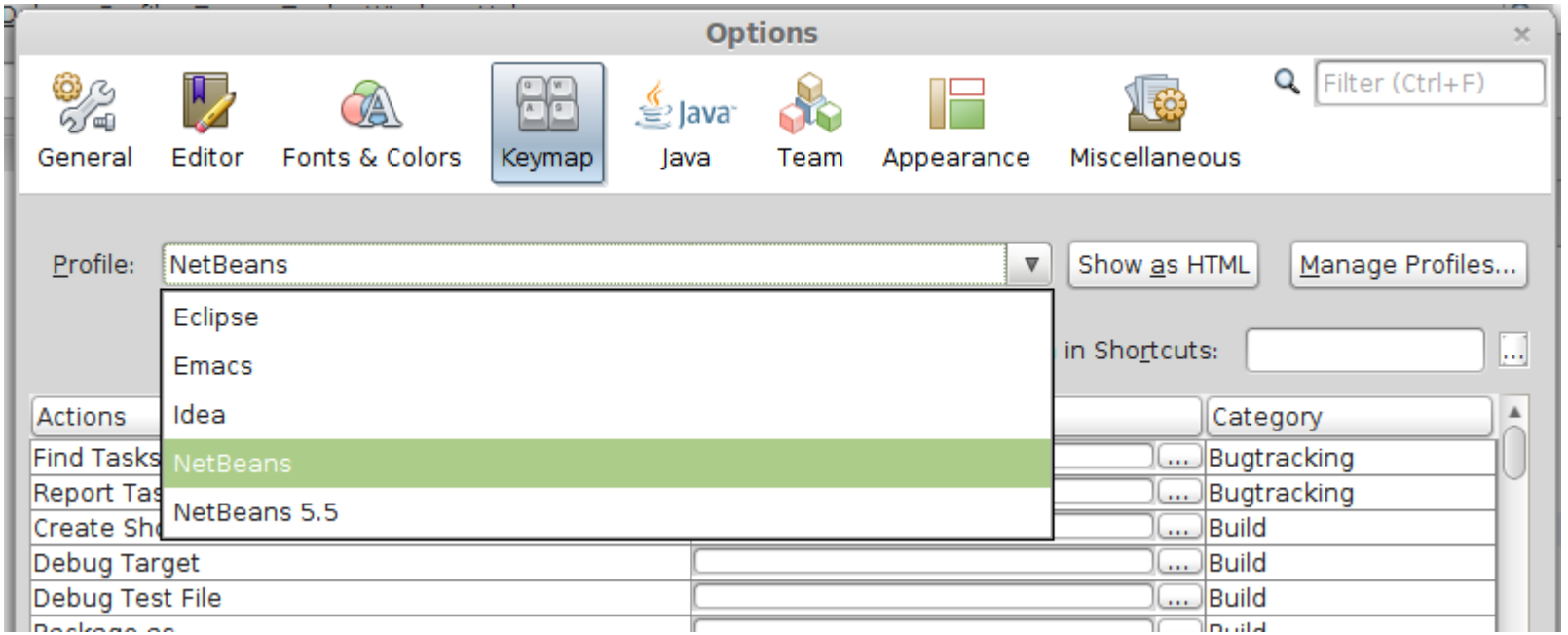
- Short exercises to reinforce the material
- Will be using NetBeans for the exercises.

Why NetBeans?

- It comes bundled with everything we need
  - Java EE compliant application server (GlassFish)
  - RDBMS (JavaDB), including a sample database
- It rocks. :)

Why NetBeans? (cont'd)

- But I'm used to Eclipse/IntelliJ IDEA!
- Tools | Options | Keymap



JavaServer Faces (JSF)

- HTML5 Friendly Markup
  - Passthrough Attributes
  - Passthrough Elements

JSF HTML Friendly Markup

- Passthrough attributes
  - Pages are built primarily using JSF components
  - Arbitrary HTML5 attributes can be passed to JSF components
  - Allow to easily adopt new HTML attributes

JSF HTML Friendly Markup (cont'd)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://xmlns.jcp.org/jsf/passthrough">
    <h:body>
        <h:form id="customerForm">
                <h:outputLabel for="firstName" value="First Name:">
                </h:outputLabel>
                <h:inputText id="firstName"
                        label="First Name"
                        value="#{customer.firstName}"
                        required="true"
                        p:placeholder="First Name">
                </h:inputText>
                <h:commandButton action="confirmation" value="Save"/>
        </h:form>
    </h:body>
</html>
```

JSF HTML Friendly Markup (cont'd)

- Sometimes we need to add more than one passthrough attribute
- Use nested <f:passThroughAttribute> JSF components

JSF HTML Friendly Markup (cont'd)

```
<h:inputText>
    <f:passThroughAttribute name="type" value="email"/>
    <f:passThroughAttribute name="placeholder"
        value="username@example.com"/>
</h:inputText>
```

JSF HTML Friendly Markup (cont'd)

- <f:passThroughAttributes>
  - Allows binding of passthrough-attributes to a managed bean property
  - Property must be of type Map<String, Object>
  - Property names are the map keys
  - Property values are the map values
  - Allows to set attributes programmatically.

JSF HTML Friendly Markup (cont'd)

```
@Named
@RequestScoped
public class PassThroughAttributesDemo{
  private Map<String, Object> attributeMap =
    new HashMap<>();
  public PassThroughAttributesDemo(){
    //initialize map here
  }
  //getter and setter here
}
```

JSF HTML Friendly Markup (cont'd)

```
<h:inputText>
  <f:passThroughAttributes  value=

“#{passThroughAttributesDemo.attributeMa
p}“/>
</h:inputText>
```

JSF HTML Friendly Markup (cont'd)

- JSF passthrough-elements
  - Applications built using HTML markup
  - Use the xmlns:jsf="http://xmlns.jcp.org/jsf namespace
  - We add JSF attributes to HTML5 elements by prefixing them with "jsf"
  - Prefixed JSF attribute is ignored by the browser, but processed by the JSF engine.
  - JSF engine converts the HTML tag into the equivalent JSF component

JSF HTML Friendly Markup (cont'd)

HTML5 Element Name	Identifying Attribute	Facelets Tag
a	jsf:action	h:commandLink
a	jsf:actionListener	h:commandLink
a	jsf:value	h:outputLink
a	jsf:outcome	h:link
body		h:body
button		h:commandButton

JSF HTML Friendly Markup (cont'd)

HTML5 Element Name	Identifying Attribute	Facelets Tag
img		h:graphicImage
input	type="button"	h:commandButton
input	type="checkbox"	h:selectBooleanCheckbox
input	type="color"	h:inputText
input	type="date"	h:inputText
input	type="datetime"	h:inputText
input	type="datetyme-local"	h:inputText
input	type="email"	h:inputText
input	type="month"	h:inputText

JSF HTML Friendly Markup (cont'd)

HTML5 Element Name	Identifying Attribute	Facelets Tag
input	type=“number“	h:inputText
input	type=“range“	h:inputText
input	type=“search“	h:inputText
input	type=“time“	h:inputText
input	type=“url“	h:inputText
input	type=“week“	h:inputText
input	type=“file“	h:inputFile
input	type=“hidden“	h:inputHidden
input	type=“password“	h:inputSecret

JSF HTML Friendly Markup (cont'd)

HTML5 Element Name	Identifying Attribute	Facelets Tag
input	type=“reset“	h:commandButton
input	type=“submit“	h:commandButton
input	type=“*“	h:inputText
label		h:outputLabel
link		h:outputStylesheet
script		h:outputScript
select	multiple=“*“	h:selectManyListbox
select		h:selectOneListbox
textarea		h:inputTextArea

Facelets Templating

- Page layout defined in a template
- Common areas such as header, footer, navigation menu, etc. defined on the template
- Template uses <ui:insert> tag to define common sections
- Template client uses <ui:define> tag inserts markup into the sections defined in the template

Facelets Templating (con'td)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:body>
        <div id="top" class="top">
            <ui:insert name="top"><h2>Welcome to our Site</h2></ui:insert>
        </div>
        <div>
            <div id="left">
                <ui:insert name="left">Left Navigation</ui:insert>
            </div>
            <div id="content" class="left_content">
                <ui:insert name="content">Content</ui:insert>
            </div>
        </div>
    </h:body>
</html>
```

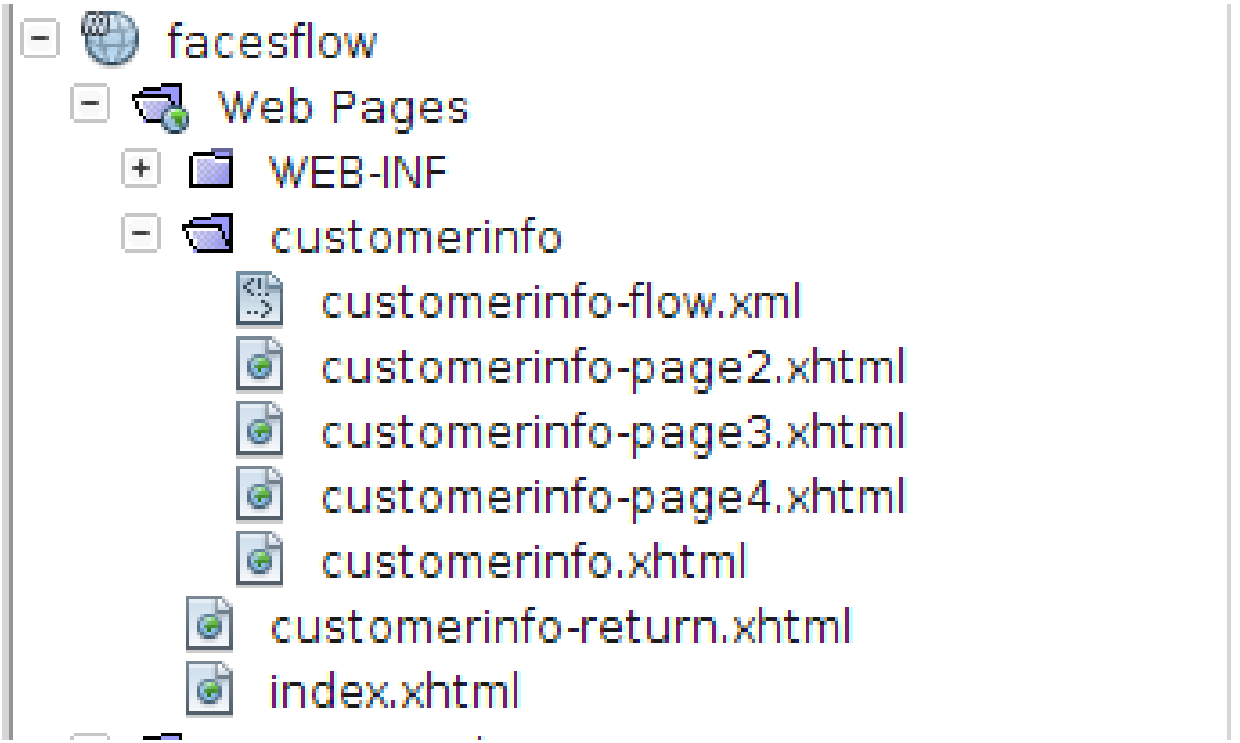
Facelets Templating (cont'd)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
    <body>
        <ui:composition template="./template.xhtml">
            <ui:define name="content">
                <p>In this area we would put our main text, images, forms, etc. </p>
            </ui:define>
            <ui:define name="left">
              <h:outputLink value="http://www.macys.com">
                <h:outputText value="Macy's"/>
              </h:outputLink>
            </ui:define>
        </ui:composition>
    </body>
</html>
```

Faces Flows

- All pages in the flow must be placed in a directory with a name that defines the name of the flow
- An XML configuration file named after the directory name and suffixed with -flow must exist inside the directory that contains the pages in the flow (the file may be empty, but it must exist)
- The first page in the flow must be named after the directory name that contains the flow
- The last page in the flow must not be located inside the directory containing the flow and must be named after the directory name and suffixed with -return

Faces Flows (cont'd)



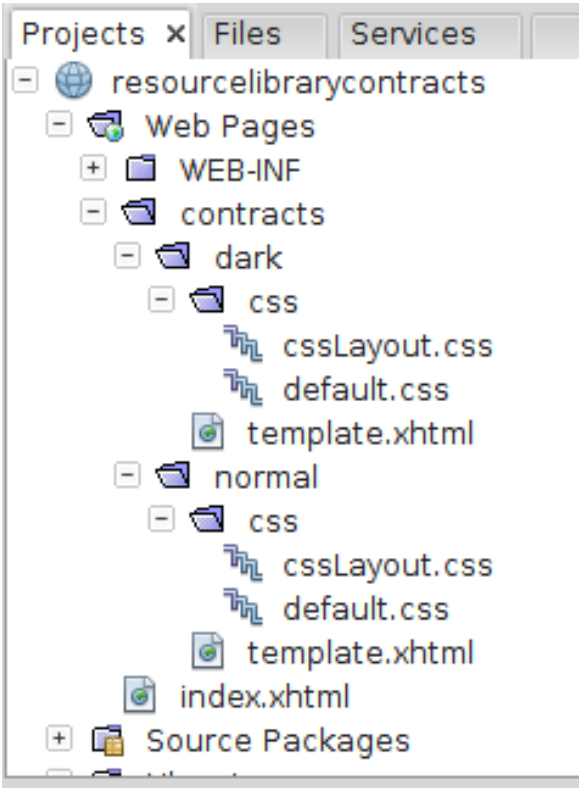
Faces Flows (cont'd)

- Bean must have a scope of flow, value attribute must match flow name

```
@Named
@FlowScoped("customerinfo")
public class Customer implements Serializable {
   //class body omitted
}
```

Resource Library Contracts

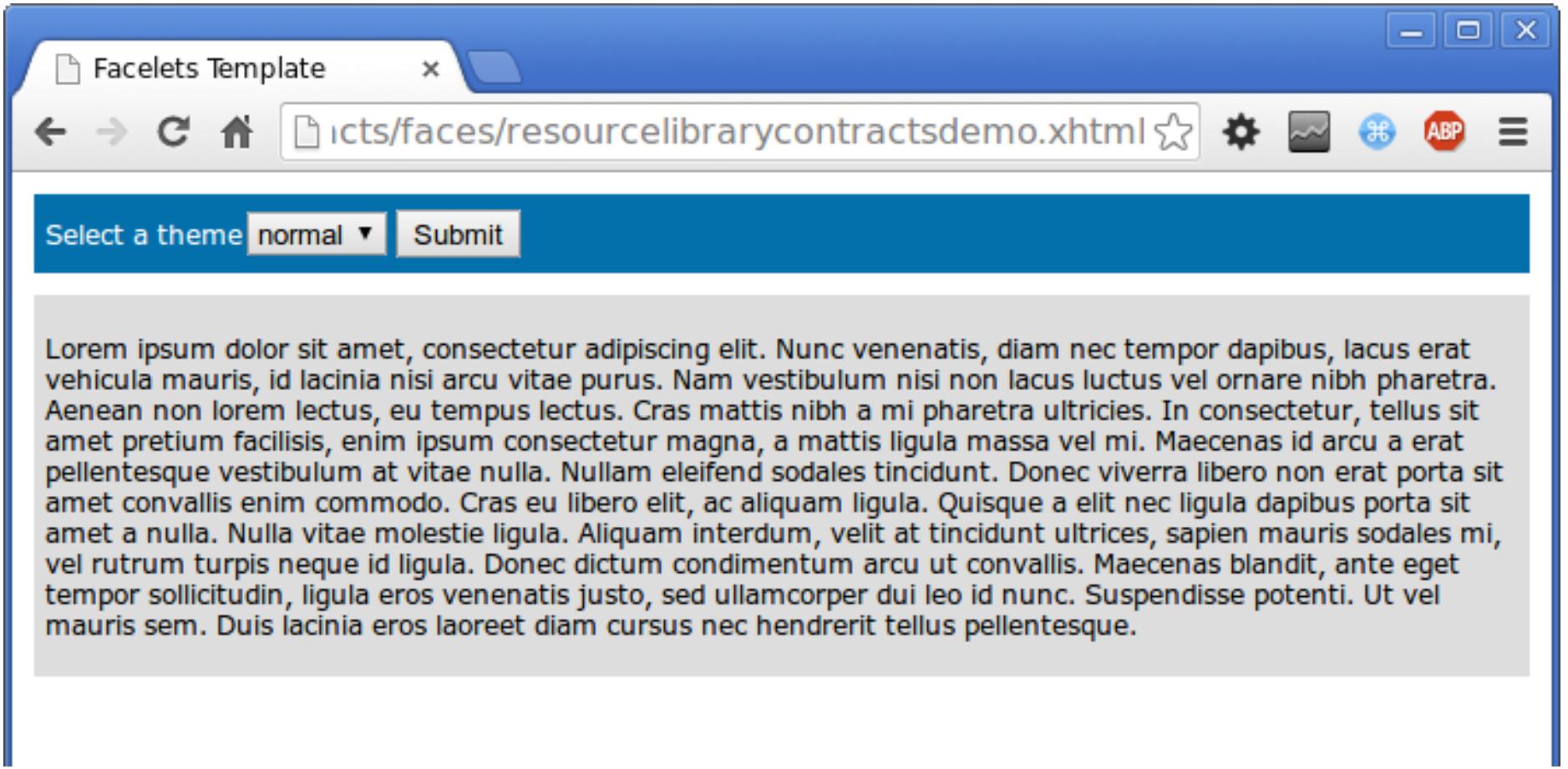
Resource Library Contracts allow themable JSF applications



Resource Library Contracts (cont'd)

```
<f:view contracts="#{themeSelector.themeName}">
        <ui:composition template="/template.xhtml">
          <ui:define name="top">
            <h:form>
              <h:outputLabel for="themeSelector"
                value="Select a theme"/>
              <h:selectOneMenu id="themeSelector"
                value="#{themeSelector.themeName}">
                <f:selectItem itemLabel="normal"
                  itemValue="normal"/>
                <f:selectItem itemLabel="dark" itemValue="dark"/>
              </h:selectOneMenu>
              <h:commandButton value="Submit"
                action="resourcelibrarycontractsdemo"/>
            </h:form>
          </ui:define>
          ...
        </ui:composition>
      </f:view>
```

Resource Library Contracts (cont'd)



Demo

- JSF features demo

Exercise 1

- Open your exercise manuals to Exercise 1

Contexts and Dependency Injection (CDI)

- Qualifiers
  - Custom annotations that allow injection of a particular implementation of a parent class or interface

CDI Qualifiers (cont'd)

```
package com.example.cdi.qualifier;

import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.inject.Qualifier;

@Qualifier
@Retention(RUNTIME)
@Target({METHOD, FIELD, PARAMETER, TYPE})
public @interface StandardCustomer {
}
```

CDI Qualifiers (cont'd)

```
package com.example.cdi.qualifier;

import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.PARAMETER;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.inject.Qualifier;

@Qualifier
@Retention(RUNTIME)
@Target({METHOD, FIELD, PARAMETER, TYPE})
public @interface PremiumCustomer {
}
```

CDI Qualifiers

```
package com.example.cdi.model;

public interface Customer {

    int getDiscountPercent();

    String getFirstName();

    String getLastName();

    void setFirstName(String firstName);

    void setLastName(String lastName);
}
```

CDI Qualifiers (cont'd)

```
package com.example.cdi.model;

import com.example.cdi.qualifier.StandardCustomer;
import javax.enterprise.context.RequestScoped;

@RequestScoped
@StandardCustomer
public class StandardCustomerBean implements Customer {

    private String firstName;
    private String lastName;

   //getters and setters omitted

    @Override
    public int getDiscountPercent() {
        return 0;
    }
}
```

CDI Qualifiers (cont'd)

```
package com.example.cdi.model;

import com.example.cdi.qualifier.PremiumCustomer;
import javax.enterprise.context.RequestScoped;

@RequestScoped
@PremiumCustomer
public class PremiumCustomerBean implements Customer {

    private String firstName;
    private String lastName;

    //getters and setters omitted

    @Override
    public int getDiscountPercent() {
        return 10;
    }
}
```

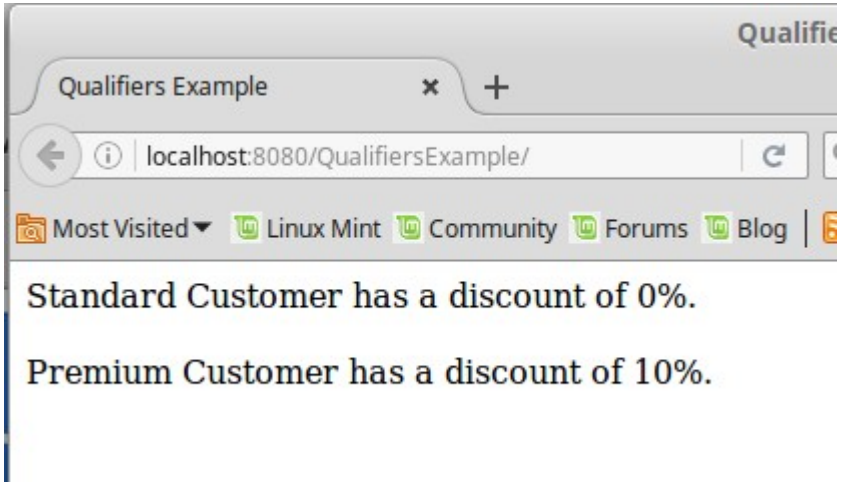
CDI Qualifiers (cont'd)

```
package com.example.cdi.controller;

//some imports omitted
import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.inject.Named;

@Named
@RequestScoped
public class CustomerController {
    @Inject
    @StandardCustomer
    private Customer standardCustomer;

    @Inject
    @PremiumCustomer
    private Customer premiumCustomer;

    //getters and setters omitted
}
```

CDI Qualifiers (cont'd)

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
   <h:head>
      <title>Qualifiers Example</title>
   </h:head>
   <h:body>
      Standard Customer has a discount of ${customerController.standardCustomer.discountPercent}%.
      <br/>
      <br/>
      Premium Customer has a discount of ${customerController.premiumCustomer.discountPercent}%.
   </h:body>
</html>
```

CDI Qualifiers (cont'd)



CDI Stereotypes

- Stereotypes
  - Custom annotations that bundle together several CDI annotations

CDI Stereotypes (cont'd)

```
package com.example.cdi.stereotype;

import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.ElementType.FIELD;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.enterprise.context.SessionScoped;
import javax.enterprise.inject.Stereotype;
import javax.inject.Named;

@Stereotype
@Named
@SessionScoped
@Retention(RUNTIME)
@Target({METHOD, FIELD, TYPE})
public @interface NamedSessionScoped {
}
```

CDI Stereotypes (cont'd)

```
package com.example.cdi.model;

import com.example.cdi.stereotype.NamedSessionScoped;
import java.io.Serializable;

@NamedSessionScoped
public class Customer implements Serializable{

    private String firstName;
    private String lastName;

    //getters and setters omitted
}
```

CDI Interceptor Binding Types

- Interceptor Binding Types
  - Allow Aspect Oriented Programming (AOP) for non-ejb CDI beans
  - Provides loose coupling between interceptors and CDI Beans

CDI Interceptor Binding Types

```
package com.example.cdi.interceptorbinding;

import static java.lang.annotation.ElementType.TYPE;
import static java.lang.annotation.ElementType.METHOD;
import static java.lang.annotation.RetentionPolicy.RUNTIME;
import java.lang.annotation.Inherited;
import java.lang.annotation.Retention;
import java.lang.annotation.Target;
import javax.interceptor.InterceptorBinding;

@InterceptorBinding
@Retention(RUNTIME)
@Target({METHOD, TYPE})
public @interface LoggingInterceptorBinding {
}
```

CDI Interceptor Binding Types (cont'd)

```
package com.example.cdi.interceptor;

@LoggingInterceptorBinding
@Interceptor
public class LoggingInterceptor implements Serializable{

    private static final Logger logger = Logger.getLogger(
            LoggingInterceptor.class.getName());

    @AroundInvoke
    public Object logMethodCall(InvocationContext invocationContext) throws Exception {

        logger.log(Level.INFO, new StringBuilder("entering ").append(invocationContext.getMethod().getName()).append( "
method").toString());
        Object retVal = invocationContext.proceed();
        logger.log(Level.INFO, new StringBuilder("leaving ").append(invocationContext.getMethod().getName()).append("
method").toString());

        return retVal;
    }
}
```

CDI Interceptor Binding Types (cont'd)

- Interceptor needs to be registered in beans.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans 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/beans_1_1.xsd"
       bean-discovery-mode="all">
    <interceptors>
     <class>
      com.example.cdi.interceptor
     </class>
    </interceptors>
</beans>
```

CDI Interceptor Binding Types (cont'd)

- Annotate CDI Bean with Interceptor Binding Type

```
package com.ensode.cdiintro.controller;
//imports omitted
@LoggingInterceptorBinding
@Named
@RequestScoped
public class PremiumCustomerController {

    private static final Logger logger = Logger.getLogger(
            PremiumCustomerController.class.getName());
    @Inject
    @Premium
    private Customer customer;

    public String saveCustomer() {
        //save customer data
    }
}
```

CDI Events

- CDI Events
  - Allow beans to communicate without any compile-time dependency.
  - Implementation of the Observer pattern

CDI Events (cont'd)

```
package com.example.cdi.controller;
import com.example.cdi.event.CustomerUpdatedEvent;
import javax.enterprise.event.Event;
//other imports omitted

@Named
@RequestScoped
public class CustomerController {

    @Inject
    @PremiumCustomer
    Event<CustomerUpdatedEvent> event;

    //additional methods and variables omitted

    public String updateCustomerData() {
        CustomerUpdatedEvent customerUpdatedEvent = new CustomerUpdatedEvent();
        customerUpdatedEvent.setCustomer(premiumCustomer);
        event.fire(customerUpdatedEvent);
        return "confirmation";
    }
}
```

CDI Events (cont'd)

```
package com.example.cdi.event;

import com.example.cdi.model.Customer;
import java.io.Serializable;

public class CustomerUpdatedEvent implements Serializable {
    private Customer customer;

    public Customer getCustomer() {
        return customer;
    }

    public void setCustomer(Customer customer) {
        this.customer = customer;
    }
}
```

CDI Events (cont'd)

```
package com.example.cdi.observer;

import com.example.cdi.event.CustomerUpdatedEvent;
import com.example.cdi.qualifier.PremiumCustomer;
import java.io.Serializable;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.enterprise.context.SessionScoped;
import javax.enterprise.event.Observes;

@SessionScoped
public class CustomerUpdateObserver implements Serializable {

    private static final Logger LOG = Logger.getLogger(CustomerUpdateObserver.class.getName());

    public void premiumCustomerUpdated(@Observes @PremiumCustomer CustomerUpdatedEvent customerUpdatedEvent) {
        LOG.log(Level.INFO, "Premium customer updated");
    }

}
```

Demo

- CDI Demo

Exercise 2

- Open your exercise manuals to Exercise 2

Enterprise JavaBeans (EJB)

- Interceptors
  - Allow Aspect-Oriented Programming (AOP)
  - Cross-cutting concerns implemented in an interceptor
  - Interceptors have a method annotated with @AroundInvoke
  - EJB to be intercepted annotated with @Interceptors

EJB 3.1 (cont'd)

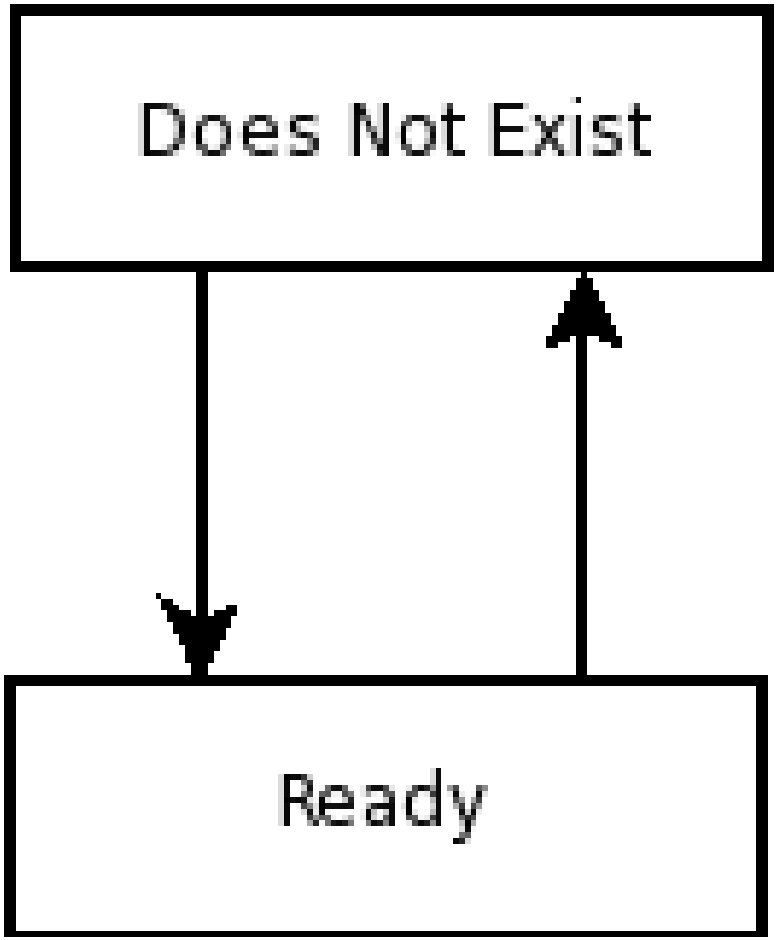
```
public class LoggingInterceptor {
    @AroundInvoke
    public Object logMethodCall(InvocationContext invocationContext) throws
Exception {
        Object interceptedObject = invocationContext.getTarget();
        Method interceptedMethod = invocationContext.getMethod();
        System.out.println("Entering " +
            interceptedObject.getClass().getName() + "." +
            interceptedMethod.getName() + "()");
        Object o = invocationContext.proceed();
        System.out.println("Leaving  "+
            interceptedObject.getClass().getName() + "." +
            interceptedMethod.getName() + "()");
        return o;
    }
}
```

EJB 3.1 (cont'd)

```
@Stateless
public class Echo {
    @Interceptors({LoggingInterceptor.class})
    public String echo(String saying) {
        return "echoing: " + saying;
    }
}
```

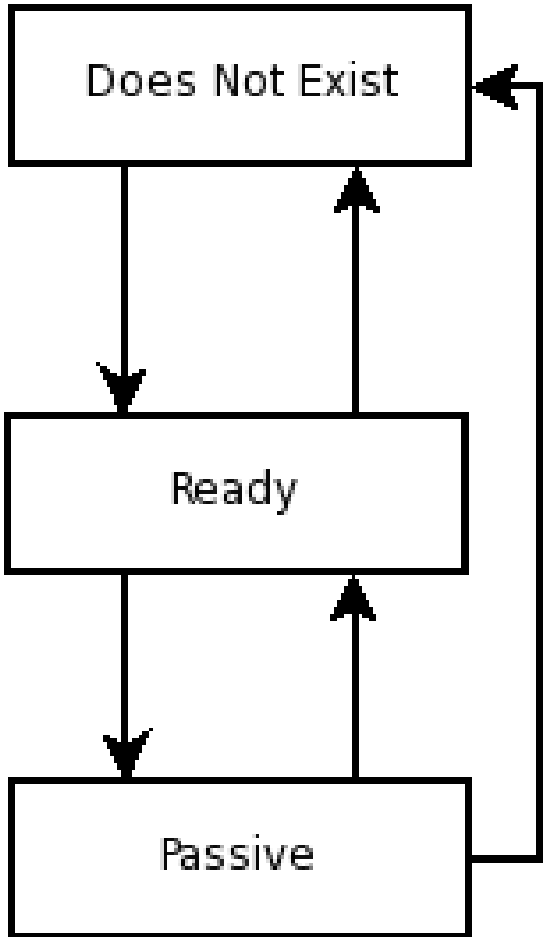
EJB Lifecycle

- Stateless/Singleton/MDB Lifecycle



EJB Lifecycle (cont'd)

- Stateful Session Bean Lifecycle



EJB Lifecycle (cont'd)

- Lifecycle annotations
  - @PostConstruct
    - Invoked when the bean is initialized.
  - @PreDestroy
    - Invoked just before the bean is destroyed

EJB Lifecycle (cont'd)

- Stateful Session Bean Lifecycle annotations
  - @PrePassivate
    - Invoked just before the bean is passivated
  - @PostActivate
    - Invoked right after the bean is activated

EJB Lifecycle (cont'd)

- All methods annotated with lifecycle annotations must be public, return void and take no arguments.

EJB Transaction Management

- Container Managed Transactions
  - By default, all EJB methods are transactional
  - Transaction starts at the beginning of the method, ends when the method exits.

EJB Transaction Management

- @TransactionAttribute
  - Allows us to define how a container managed transaction should behave if an EJB method is invoked while a transaction is in progress

EJB Transaction Management

- TransactionAttributeType.MANDATORY
  - Method must be invoked while a transaction is in progress
  - If no transaction is in progress when the method is invoked, TransactionRequiredException is thrown

EJB Transaction Management

- TransactionAttributeType.NEVER
  - Method must not be invoked while a transaction is in progress
  - If the method is invoked while a transaction is in progress, a RemoteException is thrown

EJB Container Managed Transactions

- TransactionAttributeType.NOT_SUPPORTED
  - If the method is invoked while a transaction is in progress, the transaction is suspended while the method executes, then resumed after the method finishes execution

EJB Container Managed Transactions (cont'd)

- TransactionAttributeType.REQUIRED
  - If a transaction is in progress, the method becomes part of the existing transaction
  - If no transaction is in progress, a new transaction is created
  - This is the default behavior

EJB Container Managed Transactions (cont'd)

- TransactionAttributeType.REQUIRES_NEW
  - If a transaction is in progress, it is suspended and a new transaction is created for the method. Original transaction resumes once the method completes
  - If no transaction is in progress, a new transaction is created for the method

EJB Container Managed Transactions (cont'd)

- TransactionAttributeType.SUPPORTS
  - If a transaction is in progress, the method becomes part of said transaction
  - If no transaction is in progress, no new transaction is created.

EJB Container Managed Transactions (cont'd)

```
@Stateless
@TransactionAttribute(TransactionAttributeType.
REQUIRES_NEW)
public class HelloSessionBean {
@TransactionAttribute(TransactionAttributeType.
REQUIRED)
    public String sayHello() {
        return "Hello world";
    }
}
```

EJB Asynchronous Method Invocations

- No need to wait for method invocation to return.
- When expecting a return value, a method timeout can be set

EJB Asynchronous Method Invocations (cont'd)

- Use the @Asynchronous annotation
  - Can be used at the class level or at the method level.

EJB Asynchronous Method Invocations (cont'd)

```
@Asynchronous
public void slowMethod() {
  //do time consuming stuff
}
```

EJB Asynchronous Method Invocations (cont'd)

- Client just invokes method as usual

```
public String fireAndForget() {
    asynchSessionBean.slowMethod();
    //execution continues immediately
    return "slowmethod_confirmation";
}
```

EJB Asynchronous Method Invocations (cont'd)

```
@Asynchronous
public Future<Long>
slowMethodWithReturnValue() {
  //do time consuming stuff
  //wrap return value in an instance of
  //AsynchResult
  return new AsyncResult<Long>(42L);
}
```

EJB Asynchronous Method Invocations (cont'd)

- Return value is an instance of a class implementing the Future interface using generics
- Use the Future.get() to obtain the “unwrapped” return value
- We can set a timeout on Future.get()
- Can use Future.isDone() to check if the task has completed

EJB Asynchronous Method Invocations (cont'd)

```
public String asynchReturnValue() {
  Future<Long> retVal =
  asynchSessionBean.slowMethodWithReturnValue();
  try {
    returnVal = retVal.get(5, TimeUnit.SECONDS);
  } catch (TimeoutException ex) {
    return "retval_timeout";
  } catch (InterruptedException | ExecutionException ex) {
    //handle the exception
  }
    return "retval_success";
}
```

EJB Timer Service

- Timer Service
  - Inject an instance of TimerService
  - Start the timer by invoking TimerService.createTimer()
  - Annotate the method to be invoked with @Timeout
  - Stop the timer by ivoking Timer.cancel()

EJB Timer Service (cont'd)

```
@Stateless
public class EjbTimerExampleBean
{
  @Resource
  TimerService timerService;
  ...
}
```

EJB Timer Service (cont'd)

```
public void startTimer(Serializable info)
{
  Timer timer = timerService.createTimer(
    new Date(), 5000, info);
}
```

EJB Timer Service (cont'd)

```
@Timeout
  public void logMessage(Timer timer)
  {
    logger.info("Timer expired at:"
      + timer.getInfo() + " at " +
      System.currentTimeMillis());
  }
```

EJB Timer Service (cont'd)

```
public void stopTimer(Serializable info){
    Timer timer;
    Collection timers = timerService.getTimers();
    for (Object object : timers)
         timer = ((Timer) object);
      if (timer.getInfo().equals(info)){
        timer.cancel();
        break;
      }
    }
}
```

EJB Timer Service (cont'd)

- Calendar Based Timer Expressions
  - Use the @Schedule annotation
  - Uses cron syntax
  - Timer starts automatically on application deployment

EJB Timer Service (cont'd)

```
@Stateless
public class EJBTimerDemo {
    //method is executed every 30 seconds
    @Schedule(hour = "*", minute = "*", second = "*/30")
    public void logMessage() {
        System.out.println("logMessage() method invoked at: "
                + new Date(System.currentTimeMillis()));
    }
}
```

Demo

- EJB Demo

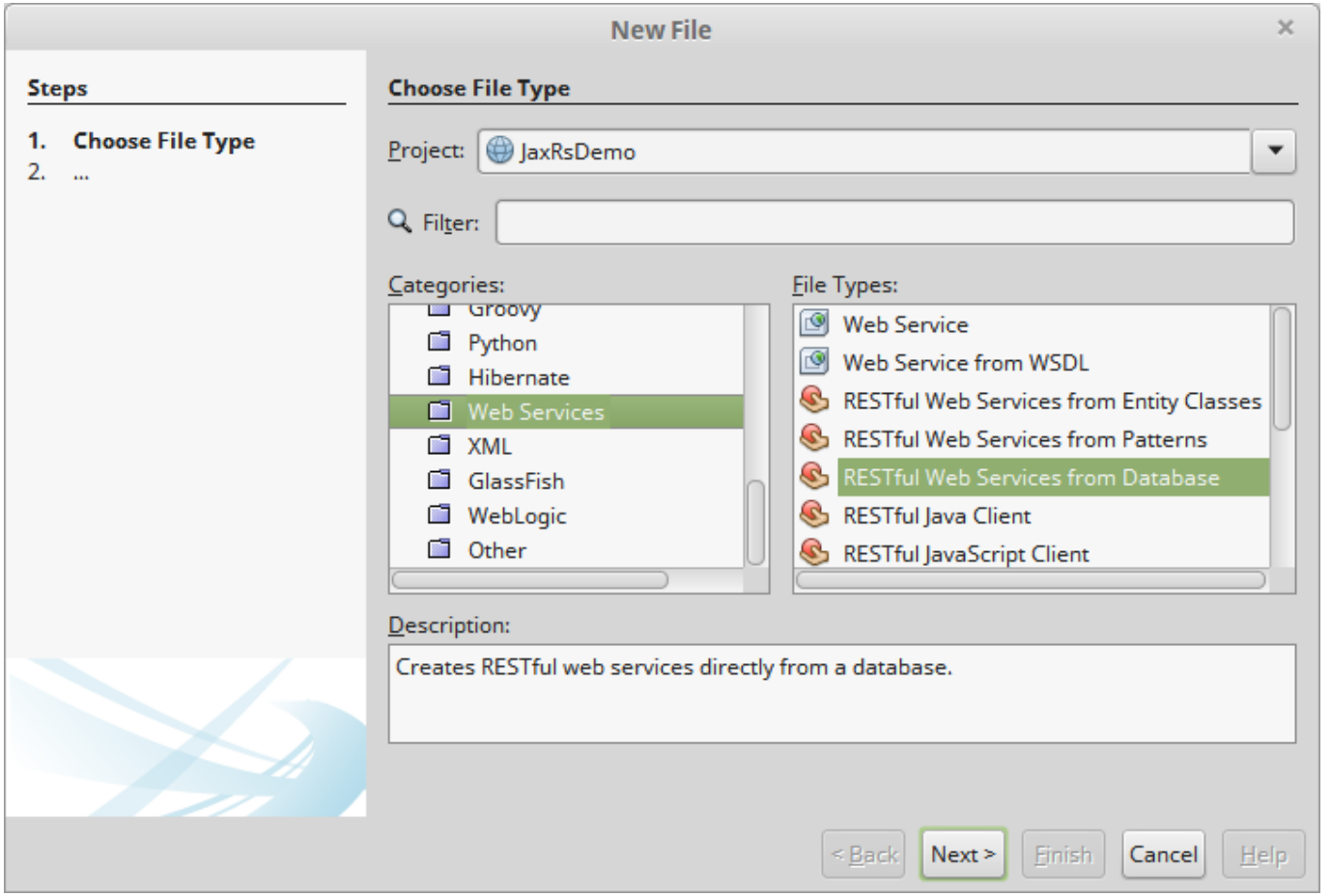
Exercise 3

- Open your exercise manuals to Exercise 3

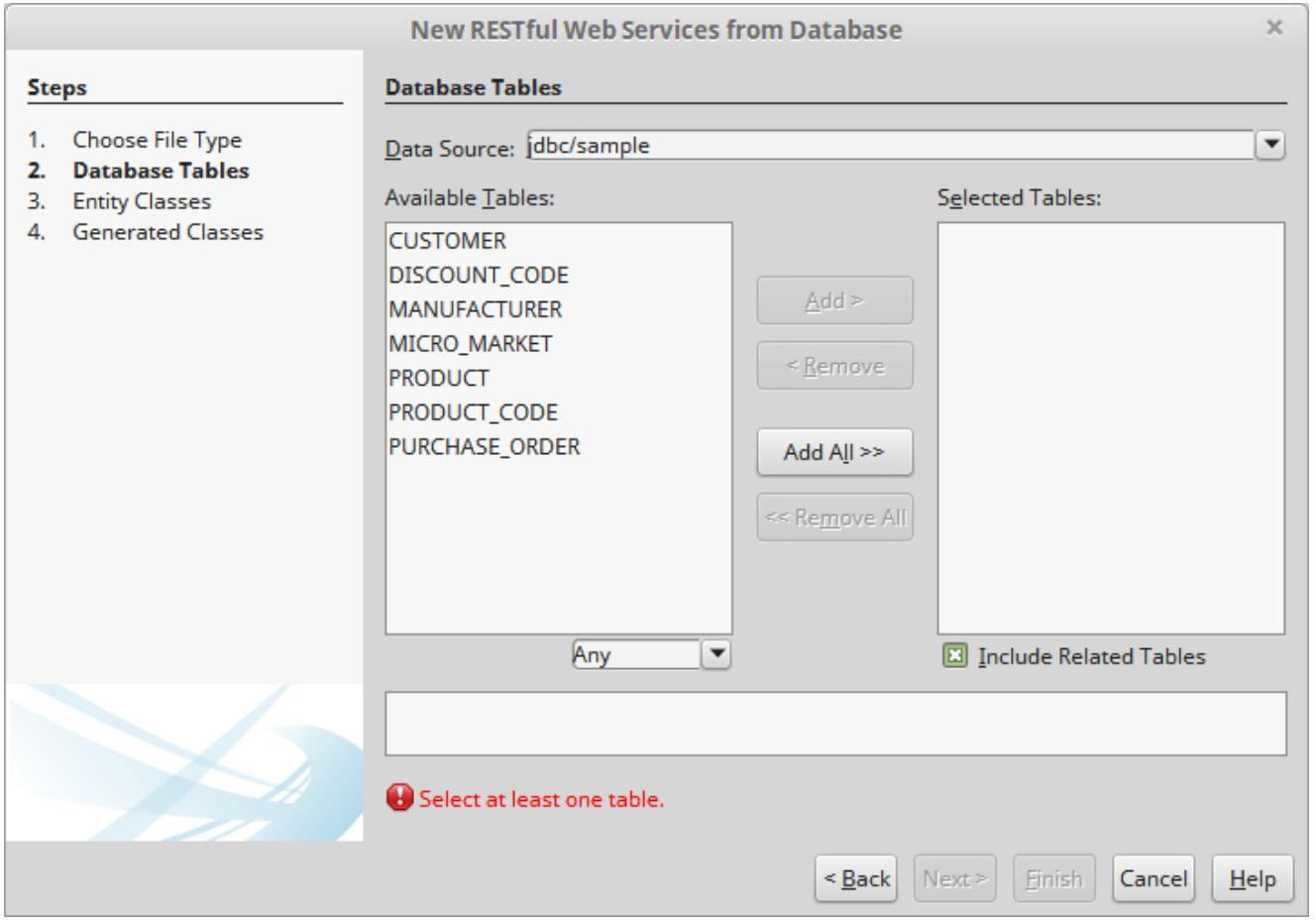
JAX-RS

- Java API for RESTful Web Services
  - Class level @Path annotation designates URI
  - Method level @GET, @POST, @PUT, @DELETE annotation to handle corresponding HTTP requests.
  - Method level @Consumes annotation designates mime type for service input (“application/xml”, “application/json”, etc)
  - Method arguments can be annotated with @QueryParam or @PathParam to handle parameters

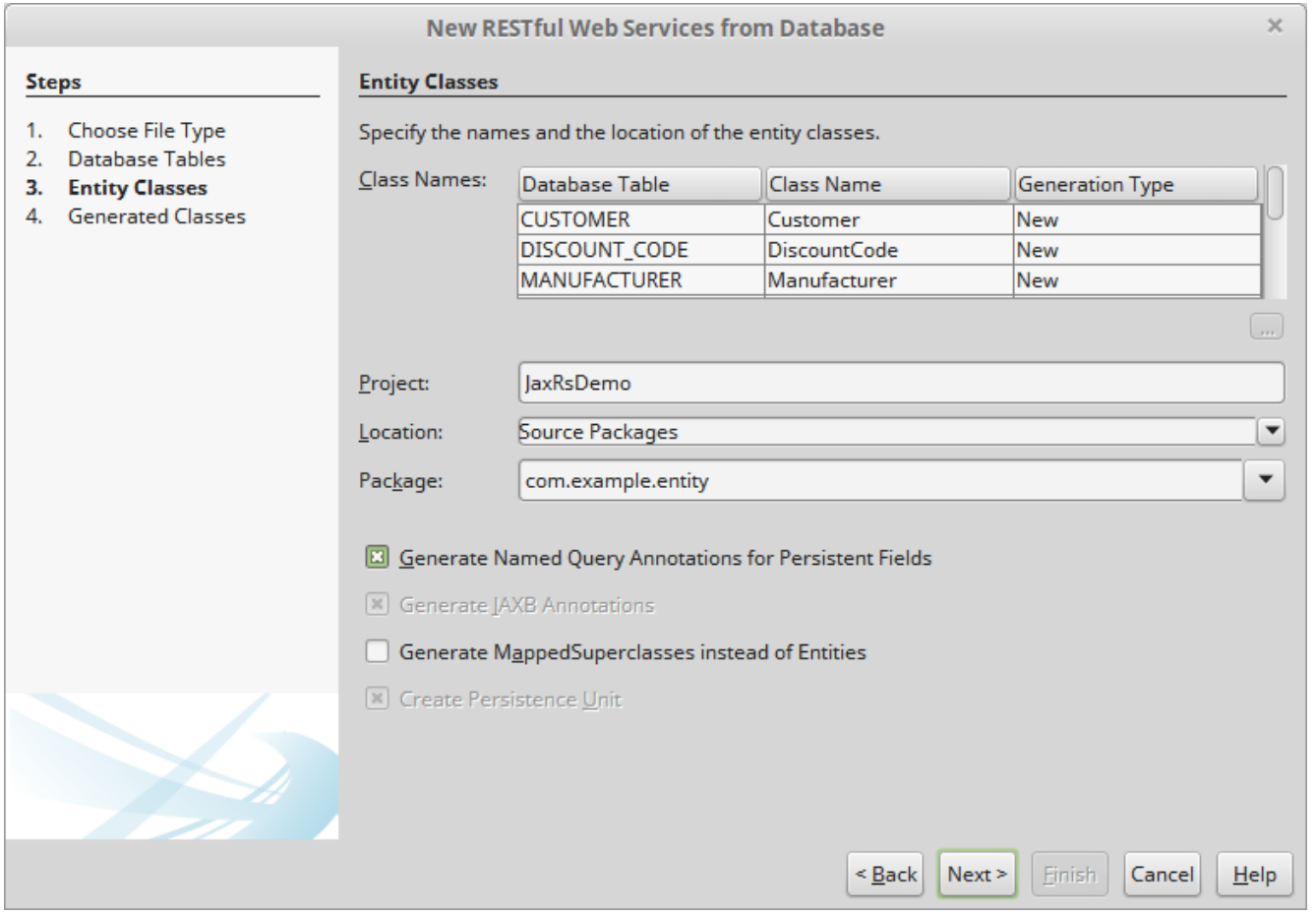
Generating JAX-RS Services from a Database



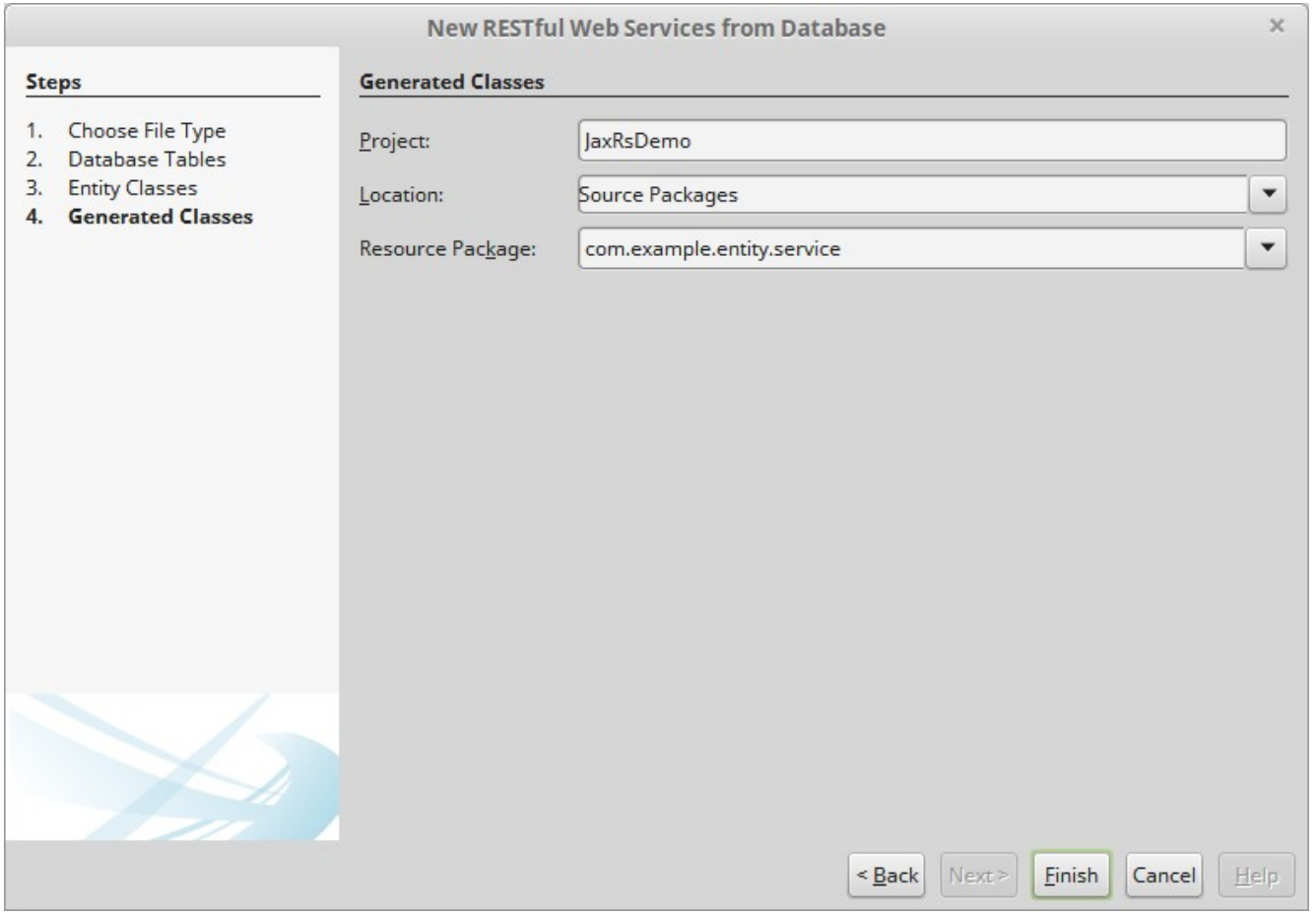
Generating JAX-RS Services from a Database



Generating JAX-RS Services from a Database



Generating JAX-RS Services from a Database



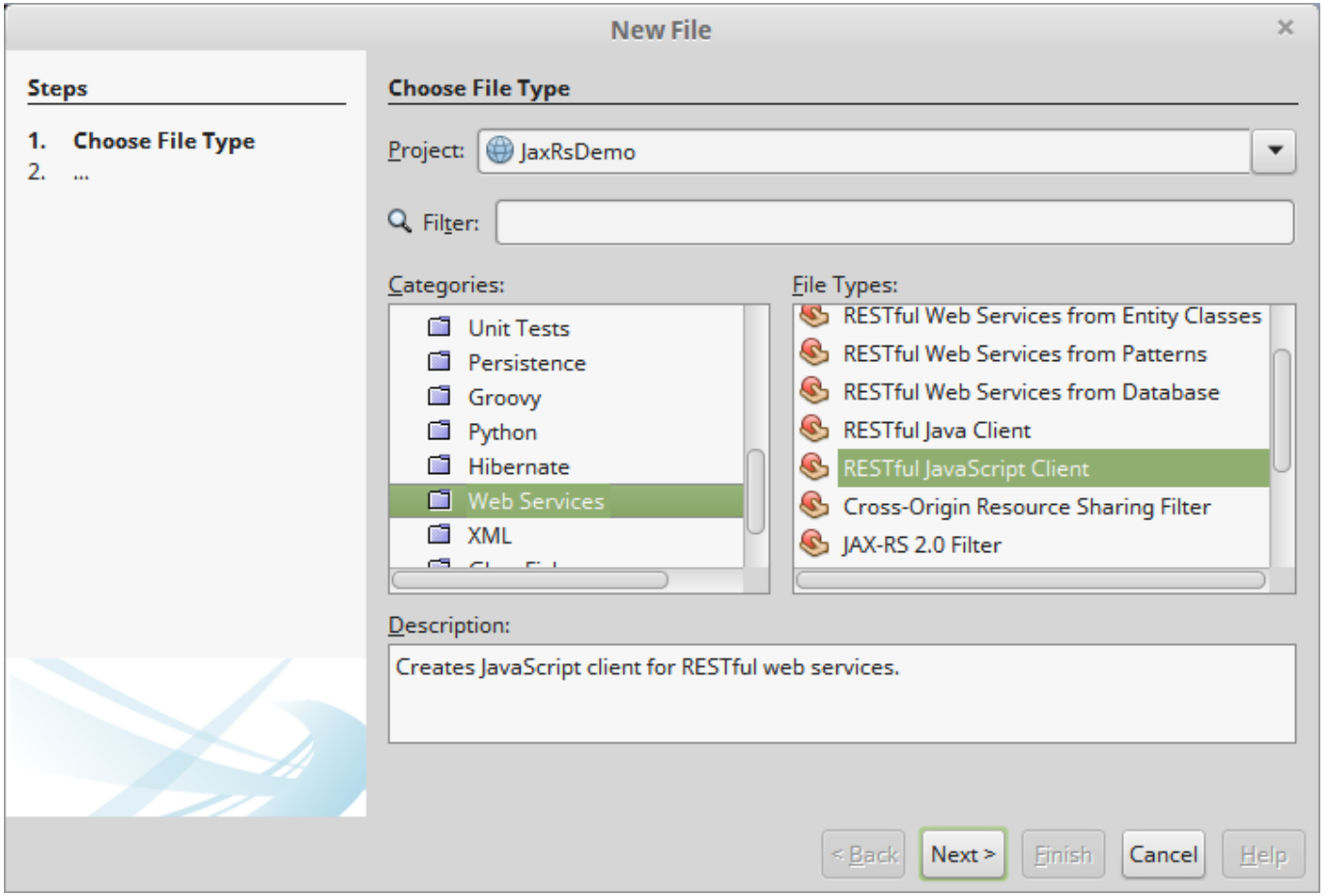
Generating JAX-RS Services from a Database

```
package com.example.entity.service;

@Stateless
@Path("com.example.entity.customer")
public class CustomerFacadeREST extends AbstractFacade<Customer> {

    @POST
    @Override
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void create(Customer entity) {
        super.create(entity);
    }

    @PUT
    @Path("{id}")
    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public void edit(@PathParam("id") Integer id, Customer entity) {
        super.edit(entity);
    }
...
```

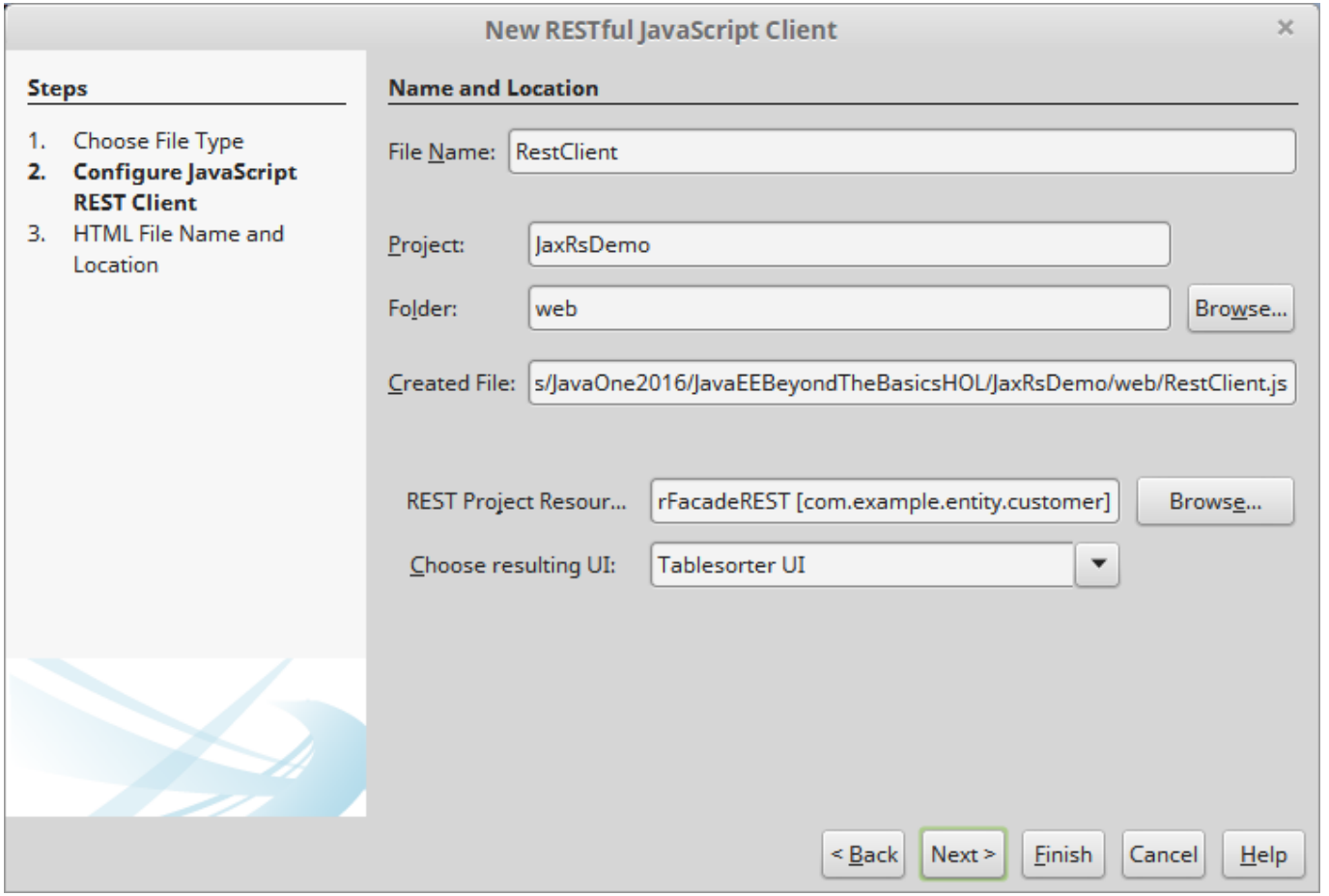
Generating JAX-RS Services from a Database

```
@DELETE
  @Path("{id}")
  public void remove(@PathParam("id") Integer id) {
      super.remove(super.find(id));
  }

  @GET
  @Path("{id}")
  @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
  public Customer find(@PathParam("id") Integer id) {
      return super.find(id);
  }
```

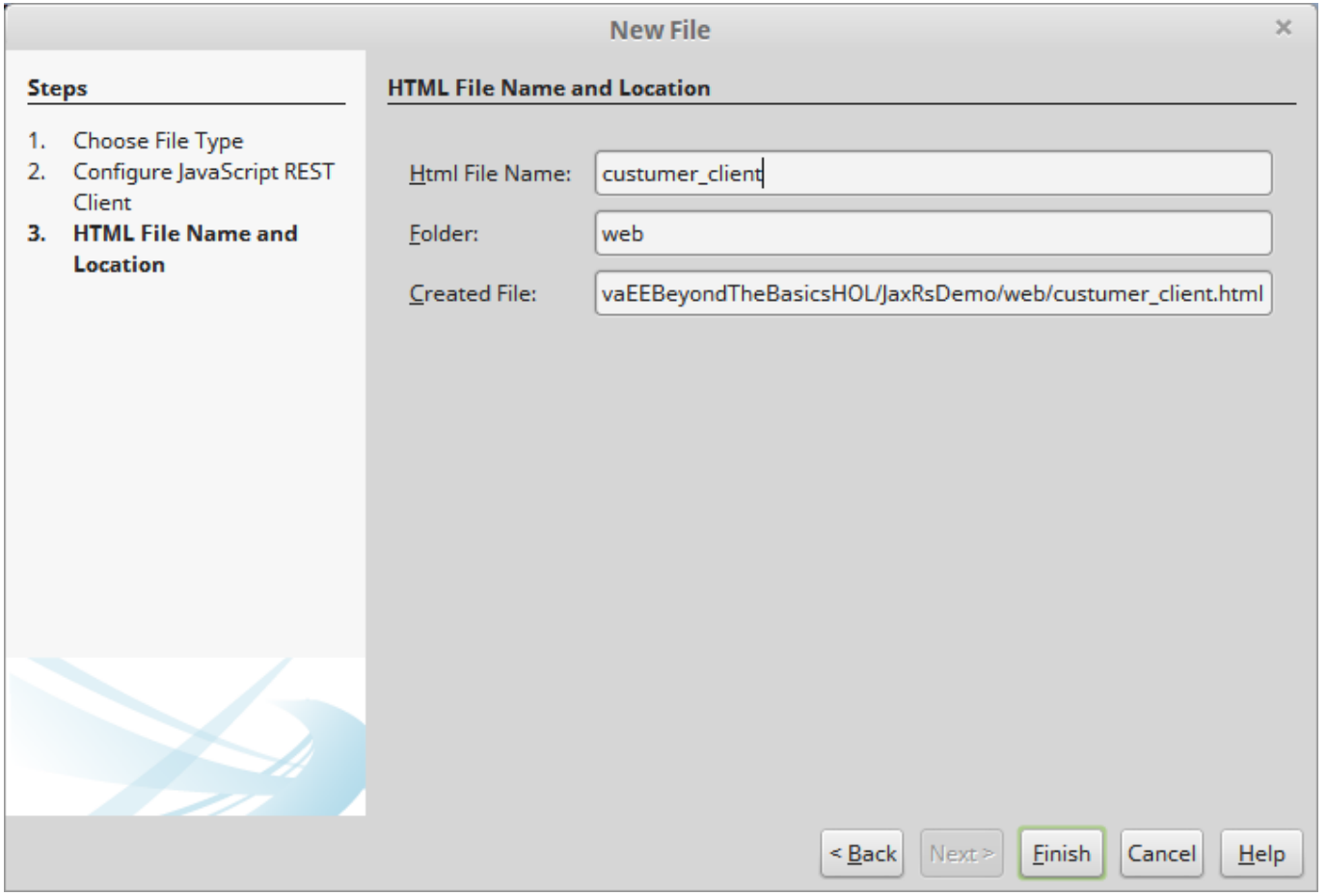
Generating JS Web Service Client



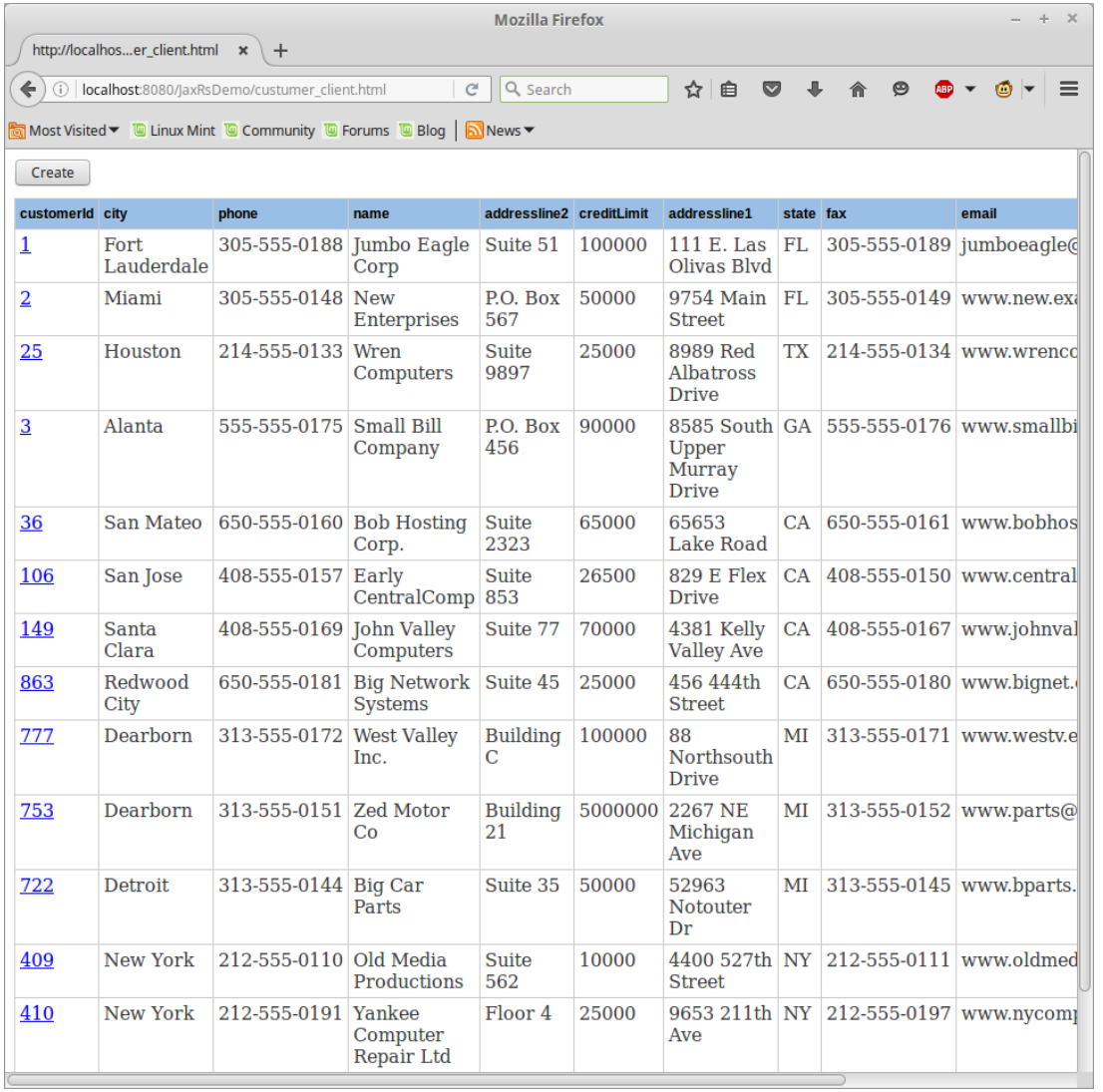
Generating JS Web Service Client



Generating JS Web Service Client



Generating JS Web Service Client



customerId	city	phone	name	addressline2	creditLimit	addressline1	state	fax	email
1	Fort Lauderdale	305-555-0188	Jumbo Eagle Corp	Suite 51	100000	111 E. Las Olivas Blvd	FL	305-555-0189	jumboeagle@
2	Miami	305-555-0148	New Enterprises	P.O. Box 567	50000	9754 Main Street	FL	305-555-0149	www.new.ex
25	Houston	214-555-0133	Wren Computers	Suite 9897	25000	8989 Red Albatross Drive	TX	214-555-0134	www.wrenc
3	Alanta	555-555-0175	Small Bill Company	P.O. Box 456	90000	8585 South Upper Murray Drive	GA	555-555-0176	www.smallbi
36	San Mateo	650-555-0160	Bob Hosting Corp.	Suite 2323	65000	65653 Lake Road	CA	650-555-0161	www.bobhos
106	San Jose	408-555-0157	Early CentralComp	Suite 853	26500	829 E Flex Drive	CA	408-555-0150	www.central
149	Santa Clara	408-555-0169	John Valley Computers	Suite 77	70000	4381 Kelly Valley Ave	CA	408-555-0167	www.johnval
863	Redwood City	650-555-0181	Big Network Systems	Suite 45	25000	456 444th Street	CA	650-555-0180	www.bignet.
777	Dearborn	313-555-0172	West Valley Inc.	Building C	100000	88 Northsouth Drive	MI	313-555-0171	www.westv.e
753	Dearborn	313-555-0151	Zed Motor Co	Building 21	5000000	2267 NE Michigan Ave	MI	313-555-0152	www.parts@
722	Detroit	313-555-0144	Big Car Parts	Suite 35	50000	52963 Notouter Dr	MI	313-555-0145	www.bparts.
409	New York	212-555-0110	Old Media Productions	Suite 562	10000	4400 527th Street	NY	212-555-0111	www.oldmed
410	New York	212-555-0191	Yankee Computer Repair Ltd	Floor 4	25000	9653 211th Ave	NY	212-555-0197	www.nycomp

Questions?

More Information