Faces Flow in JSF 2.2 – Part 2: Advanced Features

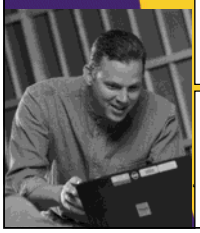
Originals of Slides and Source Code for Examples:
http://www.coreservlets.com/JSF-Tutorial/jsf2/

PRIMEFACES

JSF

For live training on JSF 2, PrimeFaces, or other Java EE topics, email hall@coreservlets.com

Marty is also available for consulting and development support



Taught by the author of *Core Servlets and JSP*, this tutorial, and JSF 2.2 version of *Core JSF*. Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - JSF 2, PrimeFaces, Ajax, jQuery, Spring MVC, JSP, Android, general Java, Java 8 lambdas/streams, GWT, custom topic mix
  - Courses available in any location worldwide. Maryland/DC area companies can also choose afternoon/evening courses.
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Hadoop, Spring, Hibernate/JPA, RESTful Web Services

Contact hall@coreservlets.com for details

Topics in This Section

- **Defining nested flows with XML**
  - Calling the nested flow
  - Sending outbound parameters from calling flow
  - Receiving inbound parameters in nested flow
- **Defining standalone flows with Java**
  - Class layout, annotations, method definition
  - Start page, views, switches, return pages
- **Defining nested flows with Java**
  - Calling nested flows, sending outbound parameters, receiving inbound parameters

Review from Faces Flow Part 1

- **Conventions**
  - Folder must contain *flowname*-flow.xml (can be empty)
  - Start page is *flowname*/*flowname*.xhtml
  - Outcomes within flow map to *flowname*/*outcome*.xhtml
  - Return page is *flowname*-return.xhtml
- **Flow-scoped beans**
  - Use @Named and @FlowScoped("*flowname*")
- **XML configuration file**
  - Custom start page: <start-node> and matching <view>
  - Return pages: <flow-return>
  - Mapping outcomes to pages: <view> & <vdl-document>
  - Conditional outcome mapping: <switch>
  - Usually use *flowname*-flow.xml, but can use faces-config

Nested Flows

Main Points

- **One flow can invoke another**
  - Calling flow uses <flow-call> in the XML configuration file (which is usually *flowname*/*flowname*-flow.xml)
- **The nested flow returns only to calling flow**
  - The <flow-return> of nested flow refers to a page of the calling flow (including return pages of calling flow)
- **Calling flow can pass data to nested flow**
  - Calling flow stores data with <outbound-parameter>
  - Nested flow receives data with <inbound-parameter>

Motivation for Nested Flows

- **Modularity**
  - Chopping self-contained part of into separate flow makes each piece easier to understand than one large flow
- **Reuse**
  - More than one flow can use the same nested flow
    - For example, two different shopping flows could use the same checkout flow
  - The calling flow just passes in the starting data
    - Each of the shopping flows might pass in the list of items to be purchased
  - The calling flow can get back a result
    - The checkout flow could return a confirmation saying whether the transaction succeeded, failed, or was canceled by the user

Calling Flow: Invoking a Nested Flow

```
<flow-call id="outcome-to-trigger-nested-flow">

  <flow-reference>
    <flow-id>id-of-the-nested-flow</flow-id>
  </flow-reference>

  <outbound-parameter>
    <name>nameOfFlowScopeParam</name>
    <value>#{bean1.someProperty}</value>
  </outbound-parameter>

  <!-- More outbound parameters -->

</flow-call>
```

E.g., nested flow can be triggered with <h:commandButton … action="outcome-to-trigger-nested-flow"/>

The id from <flow-definition> of the nested flow.

When nested flow is triggered, the system calls getSomeProperty and stores result in flowScope.nameOfFlowScopeParam. For this to be useful, the nested flow should have an inbound parameter that reads from flowScope.nameOfFlowScopeParam and stores it somewhere that is accessible in the nested flow.

Nested Flow: Getting Data From and Returning to Calling Flow

```
<flow-definition id="id-of-the-nested-flow">
  <flow-return id="outcome-to-trigger-return">
    <from-outcome>/callingflow/page</from-outcome>
  </flow-return>

  <inbound-parameter>
    <name>nameOfFlowScopeParam</name>
    <value>#{bean2.otherProperty}</value>
  </inbound-parameter>

  ...
</flow-definition>
```

The outcome in flow-return here must match an outcome in the calling flow.
E.g., it could be /callingflow/page-in-calling-flow or it could be /callingflow-return-page.

When nested flow is entered, the system looks up flowScope.nameOfFlowScopeParam and passes it to setOtherProperty.
For this to be useful, the calling flow should have an outbound parameter that stores something into flowScope.nameOfFlowScopeParam.

Matching Outbound and Inbound Parameters

- **Calling flow**

```
<outbound-parameter>
    <name>someFlowScopeParam</name>
    <value>#{beanFromCallingFlow.propertyFoo}</value>
</outbound-parameter>
```

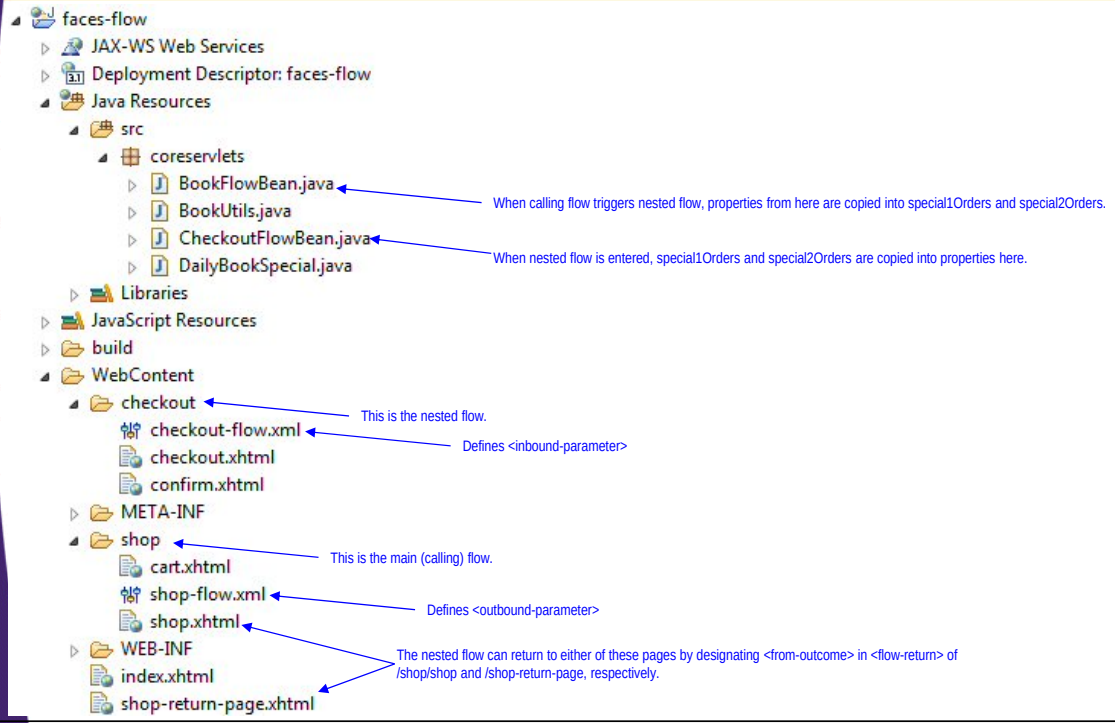
- **Nested flow**

```
<inbound-parameter>
    <name>someFlowScopeParam</name>
    <value>#{beanFromNestedFlow.propertyBar}</value>
</inbound-parameter>
```

The name of the parameter is arbitrary: what matters is that the name of the outbound parameter matches the name of the inbound parameter.

Here, when calling flow triggers the nested flow, getPropertyFoo is called and the value is stored in flowScope.someFlowScopeParam. When nested flow starts, the value of flowScope.someFlowScopeParam is passed to setPropertyBar.

Annotated Example Layout



Main Bean for Calling Flow (Part 1)

```
@Named
@FlowScoped("shop")
public class BookFlowBean implements Serializable {
  private static final long serialVersionUID = 1L;
  private DailyBookSpecial special1 = BookUtils.special1(),
                           special2 = BookUtils.special2();


  public DailyBookSpecial getSpecial1() {
    return(special1);
  }

  public DailyBookSpecial getSpecial2() {
    return(special2);
  }

  public double getTotalCost() {
    return(special1.getTotalCost() + special2.getTotalCost());
  }

  public String getTotalDollars() {
    return(BookUtils.toDollars(getTotalCost()));
  }
```

The DailyBookSpecial class stores a book title, cost, and number being ordered (initially 0).

The two specials are the books (daily specials) for sale that day. Both the main shopping flow and the nested checkout flow already know the titles and costs of the daily specials. However, the checkout (nested) flow needs to know the number of each being ordered. So, #{bookFlowBean.special1.orders} and #{bookFlowBean.special2.orders} will be passed out of the shopping flow as outbound parameters, then stored in the nested checkout flow via inbound parameters.

Main Bean for Calling Flow (Part 2)

```
public String doOrder() {
  if (getTotalCost() <= 0) {
    FacesContext context = FacesContext.getCurrentInstance();
    FacesMessage fMessage =
      new FacesMessage("You must order at least one book");
    fMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
    context.addMessage(null, fMessage);
    return(null);
  } else {
    return("cart");
  }
 }
}
```

The cart page has the link to the nested checkout flow.

DailyBookSpecial (Used by Both Flows)

```
public class DailyBookSpecial implements Serializable {
  private static final long serialVersionUID = 1L;
  private String title;
  private double price;
  private int orders;

  public DailyBookSpecial(String title, double price) {
    this.title = title;
    this.price = price;
  }

  // Getters for title and price.

  // Getters and setters for orders

  // Getters for formatted prices.

}
```

The shop flow displays two specials and lets the user set the number of orders. The checkout flow needs that information, so the number of orders are exchanged between the two flows via outbound/inbound parameters.

Main Bean for Nested Flow

```
@Named
@FlowScoped("checkout")
public class CheckoutFlowBean extends BookFlowBean {
  private static final long serialVersionUID = 1L;
  private String name, cardType, cardNumber, address, email;

  // Getters and setters for name, credit card, address, etc.

}
```

By extending BookFlowBean, the checkout bean has access to the original state (titles and costs) of the two daily specials. However, the number of each being ordered has been set in the main shopping flow, so is passed here via outbound/inbound parameters.

Configuration File for Main Flow: shop/shop-flow.xml

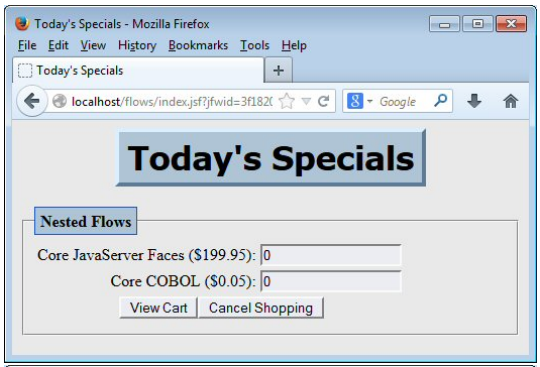
```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config ... version="2.2">
  <flow-definition id="shop">
    <flow-return id="shop-return">
      <from-outcome>/shop-return-page</from-outcome>
    </flow-return>
    <flow-call id="callCheckout">
      <flow-reference>
        <flow-id>checkout</flow-id>
      </flow-reference>
      <outbound-parameter>
        <name>special1Orders</name>
        <value>#{bookFlowBean.special1.orders}</value>
      </outbound-parameter>
      <outbound-parameter>
        <name>special2Orders</name>
        <value>#{bookFlowBean.special2.orders}</value>
      </outbound-parameter>
    </flow-call>
  </flow-definition>
</faces-config>
```

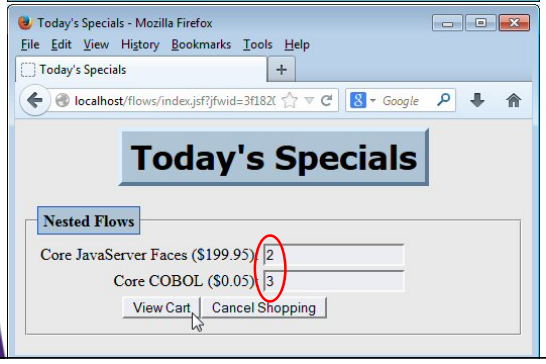
Configuration File for Nested Flow: checkout/checkout-flow.xml

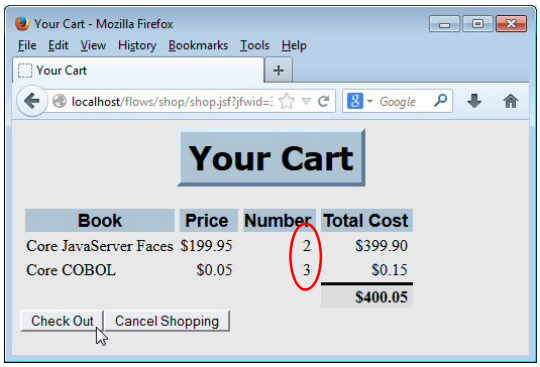
```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config ... version="2.2">
  <flow-definition id="checkout">
    <flow-return id="exit">
      <from-outcome>/shop-return-page</from-outcome>
    </flow-return>
    <flow-return id="back-to-shopping">
      <from-outcome>/shop/shop</from-outcome>
    </flow-return>
    <inbound-parameter>
      <name>special1Orders</name>
      <value>#{checkoutFlowBean.special1.orders}</value>
    </inbound-parameter>
    <inbound-parameter>
      <name>special2Orders</name>
      <value>#{checkoutFlowBean.special2.orders}</value>
    </inbound-parameter>
  </flow-definition>

</faces-config>
```


Shopping Flow



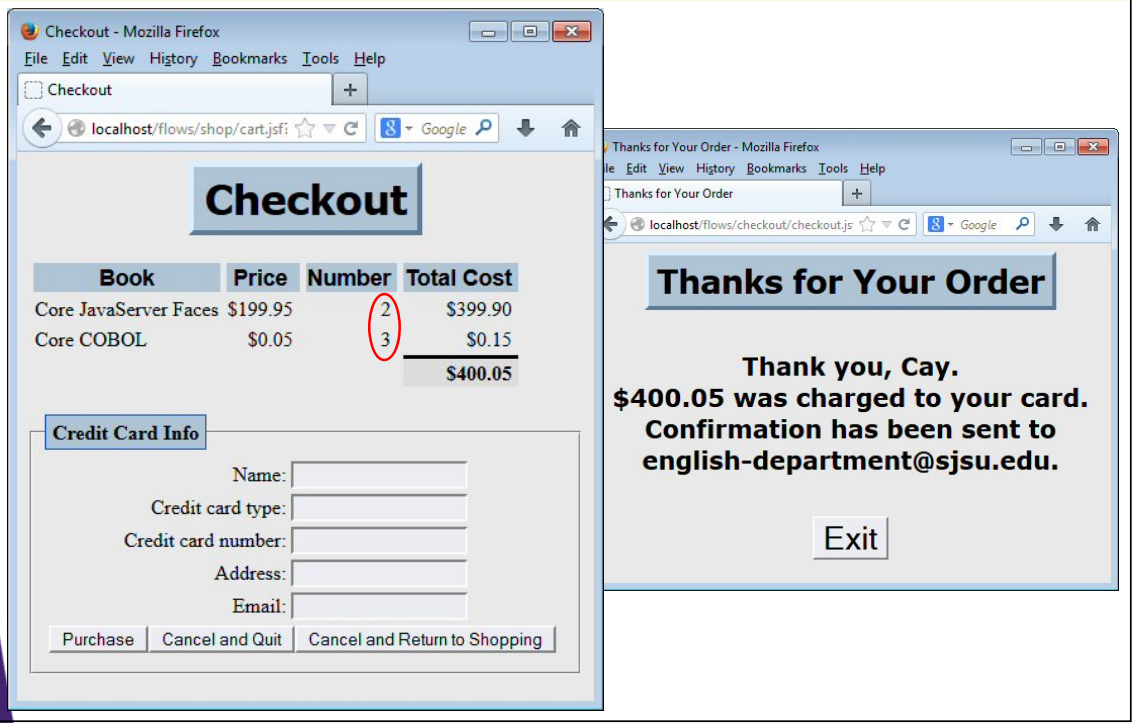




The checkout flow (next slide) already knows the titles and costs of the daily specials, but does not know the number ordered. So, those values must be sent out from the shopping flow via outbound parameters, and received in the checkout flow via inbound parameters.

Full source code for these pages, as for all examples in all sections, can be downloaded from the JSF tutorial at http://www.coreservlets.com/JSF-Tutorial/jsf2/

Checkout Flow



Defining Flows with Java: Basics

Main Points

- **You can define and configure flows with Java instead of with XML configuration file**
  - You register a Java class and method with several standard annotations. The method takes a FlowBuilder.
- **Basics in this section**
  - Define pages with builder.viewNode(…)
  - Define start page with builder.viewNode(…). markAsStartNode()
  - Define return pages with builder.returnNode(…)
  - Define switches with builder.switchNode(…)
- **More in next section**
  - Nested flows, inbound and outbound parameters

Java Flow Definition: Class Outline

```
public class MyFlowBuilder implements Serializable {
  private static final long serialVersionUID = 1L;

  @Produces
  @FlowDefinition
  public Flow defineFlow
      (@FlowBuilderParameter FlowBuilder flowBuilder) {

    // Use flowBuilder to define the flow settings

    return(flowBuilder.getFlow());
  }
}
```

Java Flow Definition: Annotated Class Outline

Despite what it says in the Java EE 7 tutorial at http://docs.oracle.com/javaee/7/tutorial/doc/jsf-configure003.htm, there is no requirement having to do with the name of the Java class. In particular, although it may be conventional to do so, it is not required that the name of the class match the flow name: that is the purpose of the id in the second argument to flowBuilder.id (shown on next slide)

```
public class MyFlowBuilder implements Serializable {
  private static final long serialVersionUID = 1L;

  @Produces
  @FlowDefinition
  public Flow defineFlow
      (@FlowBuilderParameter FlowBuilder flowBuilder) {

    // Use flowBuilder to define the flow settings

    return(flowBuilder.getFlow());
  }
}
```

It is somewhat traditional to name the method "defineFlow", but that name is not mandatory. Any method marked with the right annotations will work. You can even define multiple flows in the same class by having more than one correctly-annotated method (e.g., defineFlowA, defineFlowB).

IMPORTANT! Flow builder classes will not execute in Glassfish 4 on app startup unless WEB-INF contains beans.xml, with at least a legal start and end tag. No body content in the file is needed.

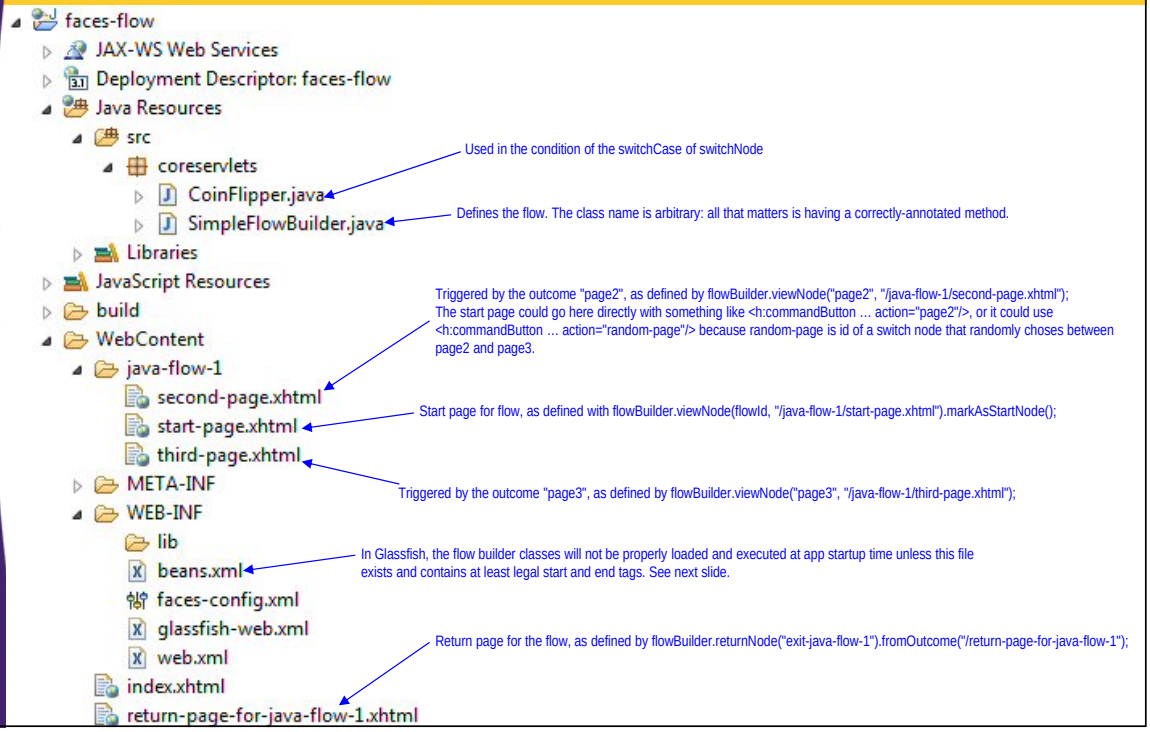
Java Flow Definition: Method Outline

The first argument to flowBuilder.id is the ID of the defining document, in case the same flow name is defined in multiple documents, as might happen in a very large project with many JAR files. In most projects, an empty String (but not null) is supplied.

The main ID of the flow (equivalent to the id attribute in <flow-definition>)

```
@Produces
@FlowDefinition
public Flow defineFlow(@FlowBuilderParameter FlowBuilder flowBuilder) {
  String flowId = "some-id";
  flowBuilder.id("", flowId);
  flowBuilder.viewNode(flowId, "/some-folder/some-page.xhtml")
             .markAsStartNode();
  flowBuilder.viewNode("other-id", "/some-folder/other-page.xhtml");
  // More view nodes
  flowBuilder.switchNode("switch-id")
             .defaultOutcome("id-defined-above-with-view-node")
             .switchCase().condition("#{some EL test}")
                          .fromOutcome("another-id-defined-above");
  // More switch nodes
  flowBuilder.returnNode("id-for-leaving-flow")
             .fromOutcome("/return-page-for-this-flow");
  // More return nodes
  return(flowBuilder.getFlow());
}
```

Annotated Example Layout



Need for beans.xml

- **Importance**
  - In Glassfish, the flow builder classes will not be properly loaded and executed at app startup time unless beans.xml exists and contains at least legal start and end tags.
    - If your Java-defined flow does not seem to be working, first find out if the flow definition class was even loaded. Put an empty constructor in your flow builder Java class, and then put a print statement or breakpoint in that default constructor. Restart the app and see if the Java class is loaded.
- **Code**

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                           http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
</beans>
```

No need to memorize this. Eclipse and NetBeans will insert it in WEB-INF automatically. And, it is included in the faces flow sample projects in the JSF tutorial at http://www.coreservlets.com/JSF-Tutorial/jsf2/

Example: Flow Definition Class

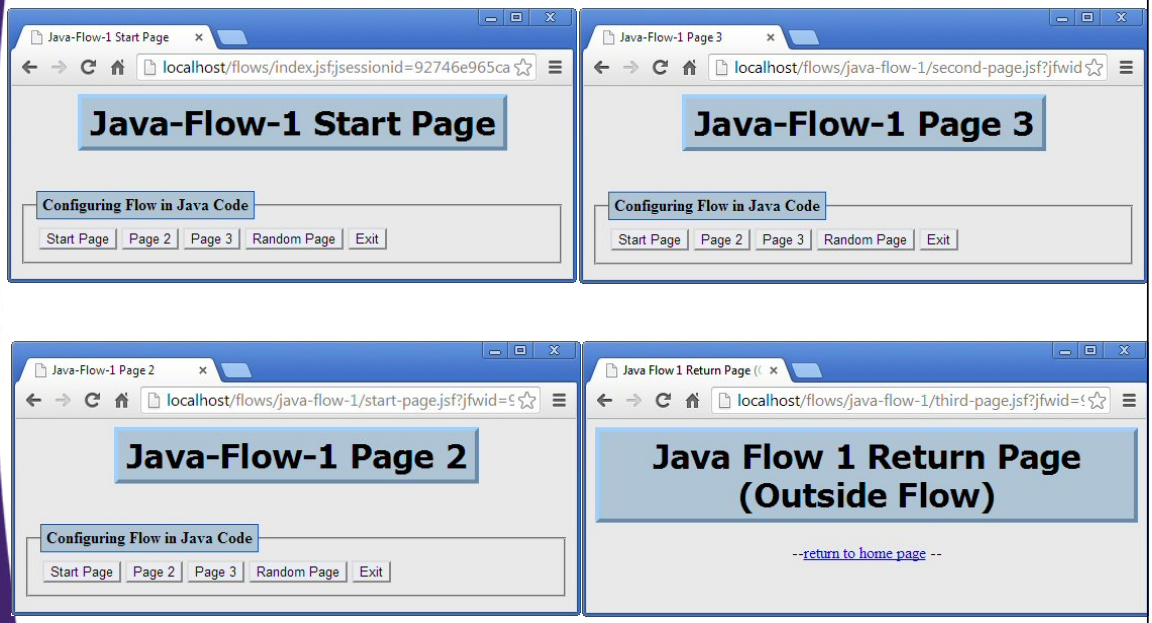
```
public class SimpleFlowBuilder implements Serializable {
  private static final long serialVersionUID = 1L;

  @Produces
  @FlowDefinition
  public Flow defineFlow(@FlowBuilderParameter FlowBuilder flowBuilder) {
    String flowId = "firstJavaFlow";
    flowBuilder.id("", flowId);
    flowBuilder.viewNode(flowId, "/java-flow-1/start-page.xhtml")
               .markAsStartNode();
    flowBuilder.viewNode("page2", "/java-flow-1/second-page.xhtml");
    flowBuilder.viewNode("page3", "/java-flow-1/third-page.xhtml");
    flowBuilder.switchNode("random-page")
               .defaultOutcome("page3")
               .switchCase().condition("#{coinFlipper.heads}")
                            .fromOutcome("page2");
    flowBuilder.returnNode("exit-java-flow-1")
               .fromOutcome("/return-page-for-java-flow-1");
    return(flowBuilder.getFlow());
  }
}
```


Example: Summary of Pages

- **Link in outside page that starts the flow**
  - <h:commandLink … action="firstJavaFlow"/>
    - "firstJavaFlow" was second argument to builder.id, and then the viewNode for that ID used markAsStartNode()
- **Link that triggers second-page.xhtml**
  - <h:commandButton ... action="page2"/>
    - Mapped via builder.viewNode
    - The mapping of "page3" to "third-page.xhtml" is similar
- **Link that triggers switch node**
  - <h:commandButton ... action="random-page"/>
    - This in turn, triggers either "page2" or "page3"
- **Link that exits flow**
  - <h:commandButton ... action="exit-java-flow-1"/>
    - Defined with builder.returnNode

Example Results



Configuring Flows with Java: Nested Flows

Main Points

- **Calling a nested flow and passing params**

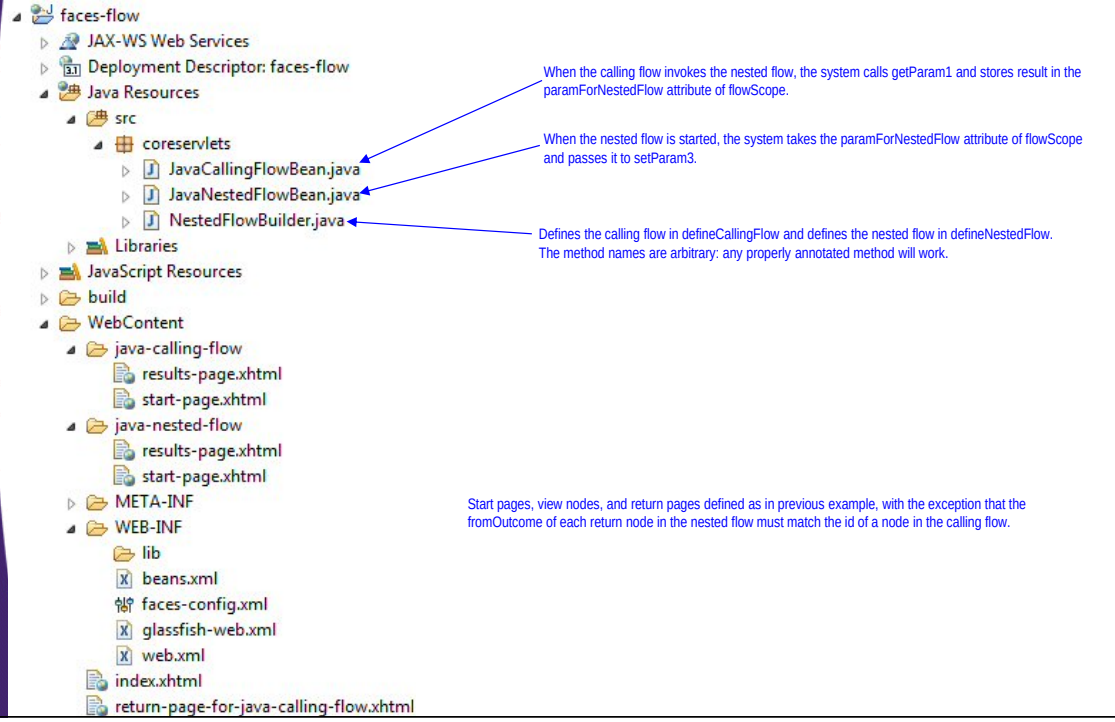
```
builder.flowCallNode("triggering-outcome")
        .flowReference("", "id-of-nested-flow")
        .outboundParameter("paramForNestedFlow",
                           "#{callingBean.property1}");
```

- **Receiving inbound parameters**

```
builder.inboundParameter("paramForNestedFlow",
                         "#{nestedBean.property2}");
```

- **Multiple flow-definition methods in class**
  - Legal to have multiple annotated methods in same class
  - With nested flows, makes sense to define both calling flow and nested flow in same class (one method for each)

Annotated Example Layout



Example: Flow Definition Class (Part 1)

```
public class NestedFlowBuilder implements Serializable {
  private static final long serialVersionUID = 1L;

  @Produces
  @FlowDefinition
  public Flow defineCallingFlow
                    (@FlowBuilderParameter FlowBuilder flowBuilder) {
    String flowId = "secondJavaFlow";
    flowBuilder.id("", flowId);
    flowBuilder.viewNode(flowId, "/java-calling-flow/start-page.xhtml")
               .markAsStartNode();
    flowBuilder.viewNode("results",
                         "/java-calling-flow/results-page.xhtml");
    flowBuilder.returnNode("return")
               .fromOutcome("/return-page-for-java-calling-flow");
    flowBuilder.returnNode("home")
               .fromOutcome("/index");
    flowBuilder.flowCallNode("go-to-nested")
               .flowReference("", "thirdJavaFlow")
               .outboundParameter("paramForNestedFlow",
                                  "#{javaCallingFlowBean.param1}");
    return(flowBuilder.getFlow());
  }
```

The outcome "go-to-nested" invokes thirdJavaFlow as a nested flow. When that nested flow is invoked, getParam1() is called, and the result is stored into flowScope.paramForNestedFlow. The thirdJavaFlow flow needs an inbound-parameter to say where flowScope.paramForNestedFlow should go.

Example: Flow Definition Class (Part 2)

```
  @Produces
  @FlowDefinition
  public Flow defineNestedFlow
                 (@FlowBuilderParameter FlowBuilder flowBuilder) {
    String flowId = "thirdJavaFlow";
    flowBuilder.id("", flowId);
    flowBuilder.viewNode(flowId,
                         "/java-nested-flow/start-page.xhtml")
               .markAsStartNode();
    flowBuilder.viewNode("results",
                         "/java-nested-flow/results-page.xhtml");
    flowBuilder.returnNode("return-to-previous-start")
               .fromOutcome("secondJavaFlow");
    flowBuilder.returnNode("return-to-previous-results")
               .fromOutcome("results");

    flowBuilder.inboundParameter("paramForNestedFlow",
                                 "#{javaNestedFlowBean.param3}");
    return(flowBuilder.getFlow());
  }
```

When this flow is invoked as a nested flow from secondJavaFlow, the value of flowScope.paramForNestedFlow is passed to the setParam3 method of javaNestedFlowBean. The paramForNestedFlow property is set via an outbound-parameter in the configuration of secondJavaFlow.

Bean for Calling Flow

```
@Named
@FlowScoped("secondJavaFlow")
public class JavaCallingFlowBean implements Serializable {
  private static final long serialVersionUID = 1L;
  private String param1="", param2="";

  // Simple getters and setters for param1 and param2:
  // getParam1, setParam1, getParam2, setParam2

  public String doFlow() {
    if (param1.equalsIgnoreCase(param2)) {
      FacesContext context = FacesContext.getCurrentInstance();
      FacesMessage fMessage =
        new FacesMessage("Params must be distinct");
      fMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
      context.addMessage(null, fMessage);
      return(null);
    } else {
      return("results");
    }
  }
}
```


Bean for Nested Flow

```
@Named
@FlowScoped("thirdJavaFlow")
public class JavaNestedFlowBean implements Serializable {
  private static final long serialVersionUID = 1L;
  private String param3, param4;

  // Simple getters and setters or param3 and param4:
  // getParam3, setParam3, getParam4, setParam4

  public String doFlow() {
    if (param3.equalsIgnoreCase(param4)) {
      FacesContext context = FacesContext.getCurrentInstance();
      FacesMessage fMessage =
        new FacesMessage("Params must be distinct");
      fMessage.setSeverity(FacesMessage.SEVERITY_ERROR);
      context.addMessage(null, fMessage);
      return(null);
    } else {
      return("results");
    }
  }
}
```

Calling Flow Start Page

```
...
<h:form>
<h:messages globalOnly="true" styleClass="error"/>
<h:panelGrid columns="3" styleClass="formTable">
  Param 1:
  <h:inputText value="#{javaCallingFlowBean.param1}" id="param1"
               required="true"
               requiredMessage="Param 1 is required"/>
  <h:message for="param1" styleClass="error"/>
  Param 2:
  <h:inputText value="#{javaCallingFlowBean.param2}" id="param2"
               required="true"
               requiredMessage="Param 2 is required"/>
  <h:message for="param2" styleClass="error"/>
  <f:facet name="footer">
    <h:commandButton value="Show Results"
                     action="#{javaCallingFlowBean.doFlow}"/><br/>
  </f:facet>
</h:panelGrid>
</h:form>
...
```


Calling Flow Results Page

```
...
<ul>
  <li>Param 1: #{javaCallingFlowBean.param1}</li>
  <li>Param 2: #{javaCallingFlowBean.param2}</li>
</ul>
<h:form>
  <h:commandButton value="Return Page" action="return"/>
  <h:commandButton value="Home Page" action="home"/>
  <h:commandButton value="Nested Flow" action="go-to-nested"/>
</h:form>
...
```

When the nested flow is invoked, the value of #{javaCallingFlowBean.param1} (i.e., the result of getParam1()) is stored into flowScope.paramForNestedFlow. The thirdJavaFlow flow has an inbound-parameter to say where flowScope.paramForNestedFlow should go.

Nested Flow Start Page

```
...
<h:form>
<h:messages globalOnly="true" styleClass="error"/>
<h:panelGrid columns="3" styleClass="formTable">
  Param 3:
  <h:inputText value="#{javaNestedFlowBean.param3}" id="param3"
               required="true"
               requiredMessage="Param 3 is required"/>
  <h:message for="param1" styleClass="error"/>
  Param 4:
  <h:inputText value="#{javaNestedFlowBean.param4}" id="param4"
               required="true"
               requiredMessage="Param 4 is required"/>
  <h:message for="param4" styleClass="error"/>
  <f:facet name="footer">
    <h:commandButton value="Show Results"
                     action="#{javaNestedFlowBean.doFlow}"/><br/>
  </f:facet>
</h:panelGrid>
</h:form>
...
```

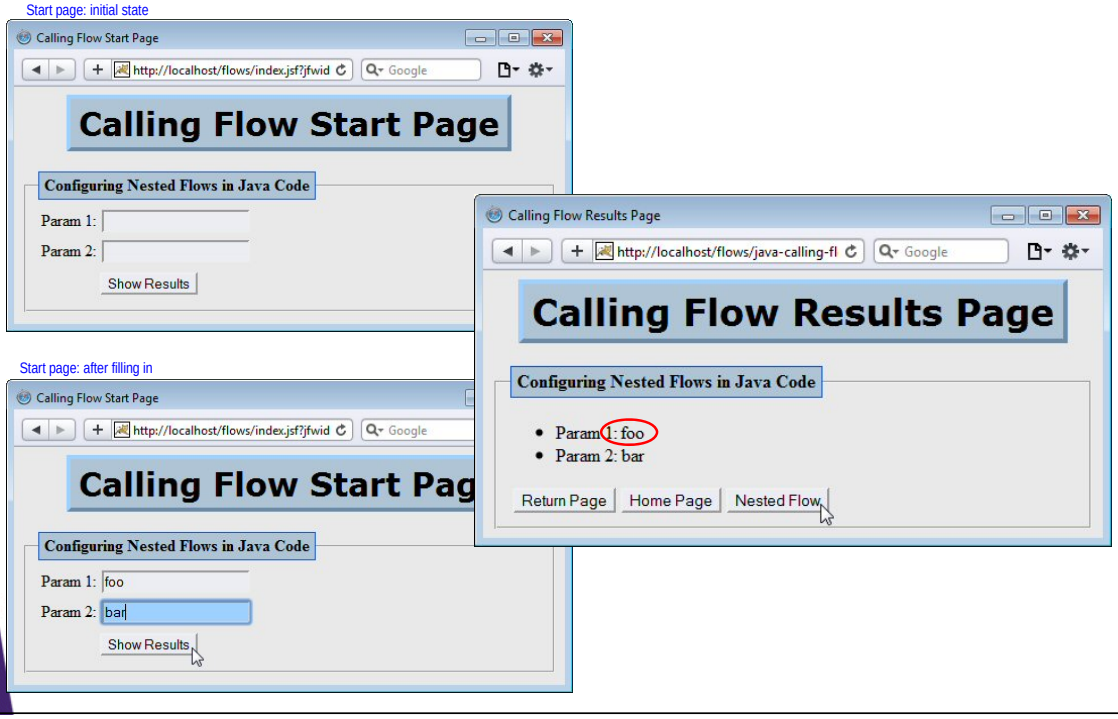
When this flow is invoked as a nested flow from secondJavaFlow, the value of flowScope.paramForNestedFlow is passed to the setParam3 method of javaNestedFlowBean. The paramForNestedFlow property was set via an outbound-parameter in the configuration of secondJavaFlow. The result is that the value of param1 in the calling flow's bean becomes the value of param3 in the nested flow's bean.

Nested Flow Results Page

```
...
<ul>
  <li>Param 3: #{javaNestedFlowBean.param3}</li>
  <li>Param 4: #{javaNestedFlowBean.param4}</li>
</ul>
<h:form>
  <h:commandButton value="Nested Flow Start Page"
                   action="thirdJavaFlow"/>
  <h:commandButton value="Calling Flow Start Page"
                   action="return-to-previous-start"/>
  <h:commandButton value="Calling Flow Results Page"
                   action="return-to-previous-results"/>
</h:form>
...
```

Results: Calling Flow

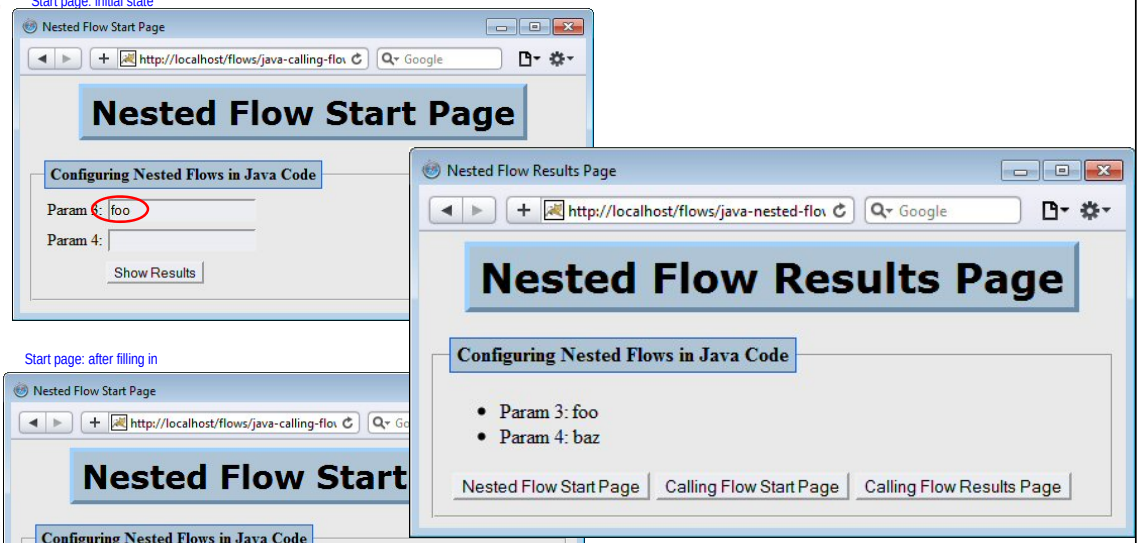
Start page: initial state



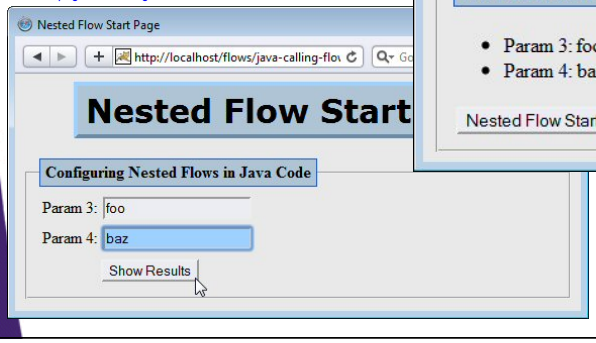
Start page: after filling in

Results: Nested Flow

Start page: initial state



Start page: after filling in



Wrap-Up

Summary

- **Nested flows configured with XML**
  - Designating flow to call: <flow-call>, <flow-reference>
  - Sending values from calling flow: <outbound-parameter>
  - Receiving values in nested flow: <inbound-parameter>
- **Defining flows with Java**
  - Outcomes & corresponding pages: builder.viewNode(…)
  - Start page: builder.viewNode(…).markAsStartNode()
  - Return pages: builder.returnNode(…)
  - Switches: builder.switchNode(…)
  - Nested flows and outbound parameters: builder.flowCallNode(…).flowReference(…).outboundParameter(…)
  - Inbound parameters: builder.inboundParameter(…)

Questions?

More info:
http://www.coreservlets.com/JSF-Tutorial/jsf2/ – JSF 2.2 tutorial
http://www.coreservlets.com/JSF-Tutorial/primefaces/ – PrimeFaces tutorial
http://courses.coreservlets.com/jsf-training.html – Customized JSF and PrimeFaces training courses
http://coreservlets.com/ – JSF 2, PrimeFaces, Java 7 or 8, Ajax, jQuery, Hadoop, RESTful Web Services, Android, HTML5, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training