COP2805C

Module 10

Java Beans

Java Persistence API (JPA)

Java Server Faces (JSF)

D. Singletary

Java Beans

- "A Java Bean is a reusable software component that can be manipulated visually in a builder tool."

  - Sun Microsystems JavaBeans API Spec

  - download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/

- Small to medium sized controls
- Can be embedded as components in applications
- Can be customized by application builders
- Uses event-based communications
- Persistent: can be customized, saved, and reloaded

3 Important Features of Beans

- The three most important features of a bean are
  1. the *properties* it exposes
  2. the set of *methods* that it allows other components to call
  3. the set of *events* that it fires

Design vs. Run-time

- A bean must be capable of running in a design environment (a builder tool)
  - It must provide information which allows customization of appearance and behavior
- A bean must be usable at run-time without the overhead of the design environment information
  - This may require separate classes

Security

- Introspection: a bean has unlimited access to introspection (determining which properties, events, and methods it supports) in a builder environment, but limited in a run-time environment
- Persistence: beans should expect to be serialized, but may have no control where the data is read from or written to
- Merging: a bean may or may not be permitted to integrate with a higher-level component

Usage Scenario

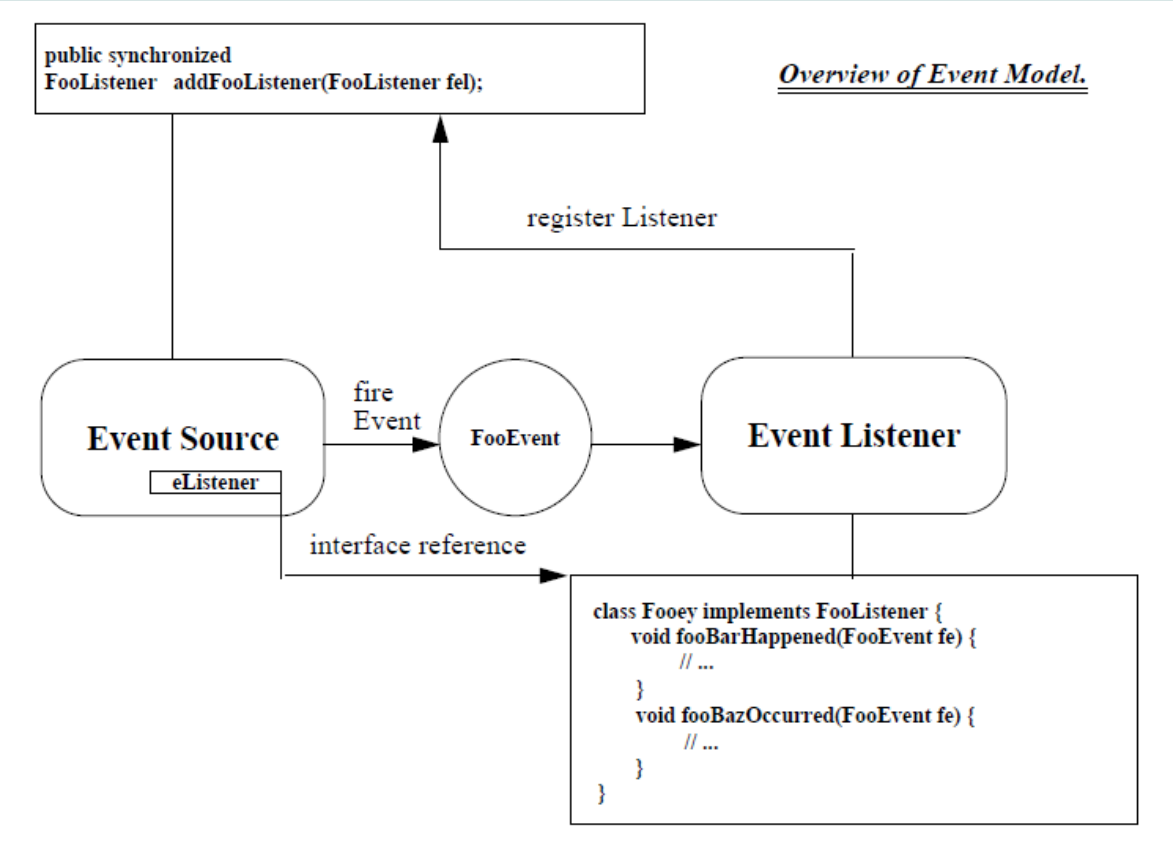
- Beans purchased/downloaded, added via **jar** file to application builder
- Application builder introspects and stores beans
- Application built with beans
- Beans are customized and connected to database via an **adapter** class
  - An adapter class provides the default implementation of all methods in an event listener interface
  - Useful when you want to process only few of the events that are handled by a particular event listener interface
  - A new class is defined by extending the adapter class and implementing relevant events
  - e.g. java.awt.event.WindowAdapter
- Completed application is packaged for release

Persistence

- All beans must support either Serialization or Externalization.
  - ("Externalization" is an option within Serialization)
- It is always valid for an application to save and restore the state of a bean using the Serialization APIs.
  - a bean should store away appropriate parts of its internal state so that it can be resurrected later with a similar appearance and behaviour.

Events

- Events are one of the core features of the Java Beans architecture.
  - Events provide a mechanism for allowing components to be plugged together in an application builder



Properties

- Properties are discrete, named attributes of a Java Bean that can affect its appearance or its behavior. Properties show up in a number of ways:
  - exposed in scripting environments as though they were fields of objects, e.g. in a Javascript environment "b.Label = foo"
  - accessed programmatically by other components calling their getter and setter methods
  - presented in a property sheet for a user to edit
- Typically a bean's properties will be persistent, so that their state will be stored away as part of the persistent state of the bean.
- Properties can have arbitrary types, including both built-in Java types such as "int" and class or interfaces types such as "Color".

Properties and Methods

- Properties are always accessed via method calls on their owning object.
  - For readable properties there will be a getter method to read the property value.
  - For writable properties there will be a setter method to allow the property value to be updated.
- Properties need not just be simple data fields; they can actually be computed values.
- Updates may have various programmatic side effects. For example, changing a bean's background color property might also cause the bean to be repainted with the new color.
- For simple properties the accessor type signatures are:
  - void setFoo(PropertyType value); // simple setter
  - PropertyType getFoo(); // simple getter

Introspection

- Introspection is enabled by implementation of the java.Beans.BeanInfo interface
  - docs.oracle.com/javase/8/docs/api/java/beans/BeanInfo.html
- Implemented methods provide descriptive information
    - getPropertyDescriptors()
    - getMethodDescriptors()
    - getEventSetDescriptors()
    - getBeanDescriptor()
    - getIcon()
- java.Beans.SimpleBeanInfo provides a base class from which customized beans can be derived

```
// BasicBean.java

import java.beans.SimpleBeanInfo;
import java.beans.PropertyDescriptor;
import java.beans.IntrospectionException;
import java.util.Arrays;

public class BasicBean extends SimpleBeanInfo {
   private static final int NUMPROPERTIES = 2;

   private String prop1 = "sidedish";
   boolean sidedish = true;

   private String prop2 = "style";
   String style = "nostyle";

   private PropertyDescriptor[] propertyDescriptors;
```

```
public BasicBean() {
    propertyDescriptors =
            new PropertyDescriptor[NUMPROPERTIES];
    try {
      propertyDescriptors[0] =
            new PropertyDescriptor(prop1, this.getClass());
      propertyDescriptors[1] =
            new PropertyDescriptor(prop2, this.getClass());
    } catch (IntrospectionException ex) {
      ex.printStackTrace();
      System.out.println(
            "failed to create property descriptors");
    }
  }
```

```
public boolean isSidedish() { return this.sidedish; }

public String getStyle() { return this.style; }

public void setSidedish(boolean sidedish) {
        this.sidedish = sidedish;
}

public void setStyle(String style) {
        this.style = style;
}

public PropertyDescriptor[] getPropertyDescriptors() {
    return propertyDescriptors;
}
```

```
    public static void main(String[] args) {
        BasicBean bb = new BasicBean();
        System.out.println(bb);
        System.out.println(Arrays.toString(bb.getPropertyDescriptors()));
    }

    public String toString() {
        return "BasicBean: style =
                " + getStyle() + ", is side = " + isSidedish();
    }

    public int hashCode() {
        return prop1.charAt(0) + prop2.charAt(0);
    }
}
```

Packaging

- Java Beans are packaged and delivered in JAR files
- A JAR file contains a contains beans with slash-separated names such as “a/b/c” ending in “.class”, e.g. “foo/bah/Aardvark.class”
  - Optionally, a serialized prototype of a bean to be used to initialize the bean. These entries must have names ending in “.ser”. E.g. “foo/elk.ser”.
  - Optional internationalization information to be used by the bean to localize itself

The Java Persistence API

- Persistence refers to storing data which outlives its creation in an application
- The Java Persistence API (JP API, or JPA) provides object/relational database mapping for Java applications
- JP API Specification
  - v2.1, 2013

Entities and Entity Operations

- Entity is a primary persistence object
  - Must be annotated with the "Entity" annotation (or via XML descriptor)
  - Must have a public or protected no-arg constructor
  - Must be a "top-level" class (ie not an inner class, enum, or interface)
  - Must not be final
- Instance variables may not be public
- Accessors and mutators must be public and must be named using conventional naming schemes,e.g. "T getX()", "void setX(T t)" where T is the type of the variable
- If entity objects are to be used remotely, must implement serializable

EntityManager

- EntityManager is a JavaEE interface which manages entity persistence contexts.

public interface javax.persistence.EntityManager

- An EntityManager instance is associated with a persistence context. A persistence context is a set of entity instances in which for any persistent entity identity there is a unique entity instance. Within the persistence context, the entity instances and their lifecycle are managed. The EntityManager API is used to create and remove persistent entity instances, to find entities by their primary key, and to query over entities.
  - ie. create/delete/modify DB objects and query them.

Entity Example

```
@Entity
public class Customer implements Serializable {
    private Long id;
    private String name;
    private Address address;

    // No-arg constructor
    public Customer() {}

    @Id // property access is used
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Address getAddress() {
        return address;
    }

    public void setAddress(Address address) {
        this.address = address;
    }
}
```

Primary Keys

- Entities must have a primary key, indicated by the "Id" annotation
- Can use a single persistent field or a composite key (class)

```
@Entity
public class Employee {
    @Id long empId;
    String empName;
    ...
}
```

```
@Embeddable
public class EmployeeId {
    String firstName;
    String lastName;
...
}
@Entity
public class Employee {
    @EmbeddedId EmployeeId empId;
    ...
}
```

Sample Query:

SELECT e
FROM Employee e
WHERE e.empId.firstName = 'Sam'

Relationships

- Relationships require annotations based on their type:
  - OneToOne
  - OneToMany
  - ManyToOne
  - ManyToMany

```
@Entity
public class Employee {
    private Cubicle assignedCubicle;
    @OneToOne
    public Cubicle getAssignedCubicle() {
        return assignedCubicle;
    }
    public void setAssignedCubicle(Cubicle cubicle) {
        this.assignedCubicle = cubicle;
    }
...

@Entity
public class Cubicle {
    private Employee residentEmployee;
    @OneToOne(mappedBy="assignedCubicle")
    public Employee getResidentEmployee() {
        return residentEmployee;
    }
    public void setResidentEmployee(Employee employee) {
        this.residentEmployee = employee;
    }
    ...
}
```

- Another Example: Customer class implemented from NetBeans CRUD (Create, Read, Update, Delete) tutorial:

```
@Entity
@Table(name = "CUSTOMER")
@XmlRootElement
@NamedQueries({
  @NamedQuery(name = "Customer.findAll", query = "SELECT c FROM Customer c")
  , @NamedQuery(name = "Customer.findByCustomerId", query = "SELECT c FROM Customer c WHERE c.customerId = :customerId")
  , @NamedQuery(name = "Customer.findByName", query = "SELECT c FROM Customer c WHERE c.name = :name")
  , @NamedQuery(name = "Customer.findByAddressline1", query = "SELECT c FROM Customer c WHERE c.addressline1 = :addressline1")
  , @NamedQuery(name = "Customer.findByAddressline2", query = "SELECT c FROM Customer c WHERE c.addressline2 = :addressline2")
  , @NamedQuery(name = "Customer.findByCity", query = "SELECT c FROM Customer c WHERE c.city = :city")
  , @NamedQuery(name = "Customer.findByState", query = "SELECT c FROM Customer c WHERE c.state = :state")
  , @NamedQuery(name = "Customer.findByPhone", query = "SELECT c FROM Customer c WHERE c.phone = :phone")
  , @NamedQuery(name = "Customer.findByFax", query = "SELECT c FROM Customer c WHERE c.fax = :fax")
  , @NamedQuery(name = "Customer.findByEmail", query = "SELECT c FROM Customer c WHERE c.email = :email")
  , @NamedQuery(name = "Customer.findByCreditLimit", query = "SELECT c FROM Customer c WHERE c.creditLimit = :creditLimit")})

public class Customer implements Serializable {
```

```
private static final long serialVersionUID = 1L;
@Id
@Basic(optional = false)
@Column(name = "CUSTOMER_ID")
private Integer customerId;
@Column(name = "NAME")
private String name;
@Column(name = "ADDRESSLINE1")
private String addressline1;
@Column(name = "ADDRESSLINE2")
private String addressline2;
@Column(name = "CITY")
private String city;
@Column(name = "STATE")
private String state;
@Column(name = "PHONE")
private String phone;
@Column(name = "FAX")
private String fax;
@Column(name = "EMAIL")
private String email;
@Column(name = "CREDIT_LIMIT")
private Integer creditLimit;
@JoinColumn(name = "DISCOUNT_CODE", referencedColumnName = "DISCOUNT_CODE")
@ManyToOne(optional = false)
private DiscountCode discountCode;
@JoinColumn(name = "ZIP", referencedColumnName = "ZIP_CODE")
@ManyToOne(optional = false)
private MicroMarket zip;

public Customer() {
}
```

```
public Customer(Integer customerId) {
    this.customerId = customerId;
  }

  public Integer getCustomerId() {
    return customerId;
  }

  public void setCustomerId(Integer customerId) {
    this.customerId = customerId;
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public String getAddressline1() {
    return addressline1;
  }

  public void setAddressline1(String addressline1) {
    this.addressline1 = addressline1;
  }
```

```
public String getAddressline2() {
   return addressline2;
}

public void setAddressline2(String addressline2) {
   this.addressline2 = addressline2;
}

public String getCity() {
   return city;
}

public void setCity(String city) {
   this.city = city;
}

public String getState() {
   return state;
}

public void setState(String state) {
   this.state = state;
}

public String getPhone() {
   return phone;
}

public void setPhone(String phone) {
   this.phone = phone;
}
```

```
public String getFax() {
   return fax;
}

public void setFax(String fax) {
   this.fax = fax;
}

public String getEmail() {
   return email;
}

public void setEmail(String email) {
   this.email = email;
}

public Integer getCreditLimit() {
   return creditLimit;
}

public void setCreditLimit(Integer creditLimit) {
   this.creditLimit = creditLimit;
}

public DiscountCode getDiscountCode() {
   return discountCode;
}

public void setDiscountCode(DiscountCode discountCode) {
   this.discountCode = discountCode;
}
```

```
    public MicroMarket getZip() {
        return zip;
    }

    public void setZip(MicroMarket zip) {
        this.zip = zip;
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (customerId != null ? customerId.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        // TODO: Warning - this method won't work in the case the id fields are not set
        if (!(object instanceof Customer)) {
            return false;
        }
        Customer other = (Customer) object;
        if ((this.customerId == null && other.customerId != null) || (this.customerId != null && !this.customerId.equals(other.customerId))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "demo.Customer[ customerId=" + customerId + " ]";
    }
}
```

Factories

- A Factory is a design pattern used to create customized instances of a class
- EntityManagerFactory is an interface used to obtain EntityManager instances

```
public final class CustomerViewerTopComponent extends TopComponent {
    public CustomerViewerTopComponent() {
        initComponents();
        setName(Bundle.CTL_CustomerViewerTopComponent());
        setToolTipText(Bundle.HINT_CustomerViewerTopComponent());
        EntityManager entityManager =
            Persistence.createEntityManagerFactory
                ("CustomerLibraryPU").createEntityManager();
        Query query =
            entityManager.createNamedQuery("Customer.findAll");
        List<Customer> resultList = query.getResultList();
        for (Customer c : resultList) {
            jTextArea1.append(c.getName() + " (" + c.getCity() + ")" + "\n");
    }
}
```

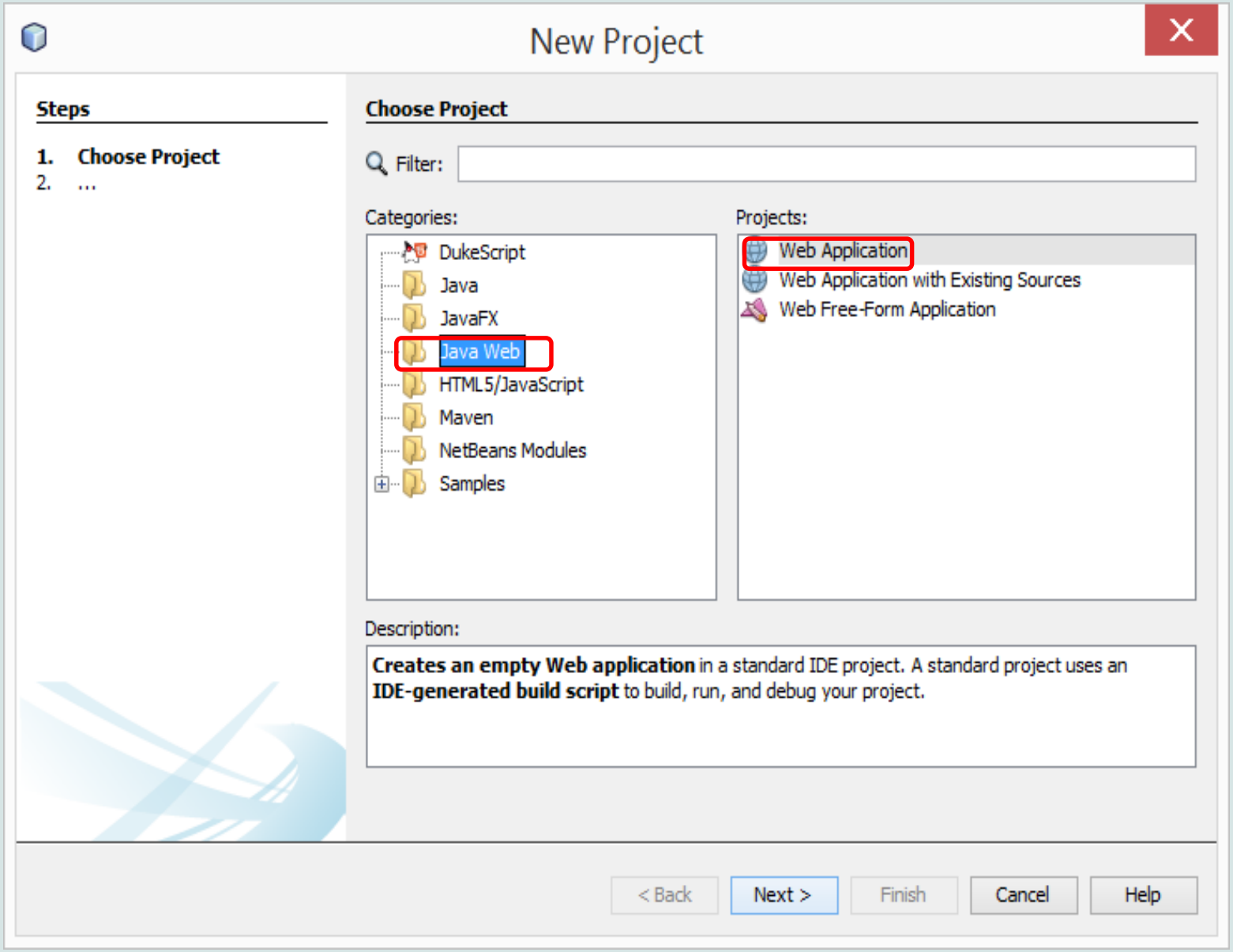
JPA Implementations

- Hibernate: hibernate.org/orm/
- Toplink:Reference Implementation of JPA 1.0, integrated into EclipseLink
- EclipseLink: eclipse.org/eclipselink/
- Apache OpenJPA: openjpa.apache.org/
- DataNucleus: datanucleus.com/
- ObjectDB: objectdb.com/

Ch. 33 JavaServer Faces (JSF)

- JSF enables you to quickly build Web applications by
  - assembling reusable UI components in a page
  - connecting the components to Java programs
  - wiring client-generated events to server-side event handlers
- Uses XHTML (eXtensible HyperText Markup Language) and CSS (Cascading Style Sheets)
  - XHTML is HTML "redesigned" as XML

Creating a JSF Project in NetBeans



New Web Application

Steps

1. Choose Project
2. **Name and Location**
3. Server and Settings
4. Frameworks

Name and Location

Project Name: jsf2demo

Project Location: F:\fscj\jtech\netbeans [Browse...]

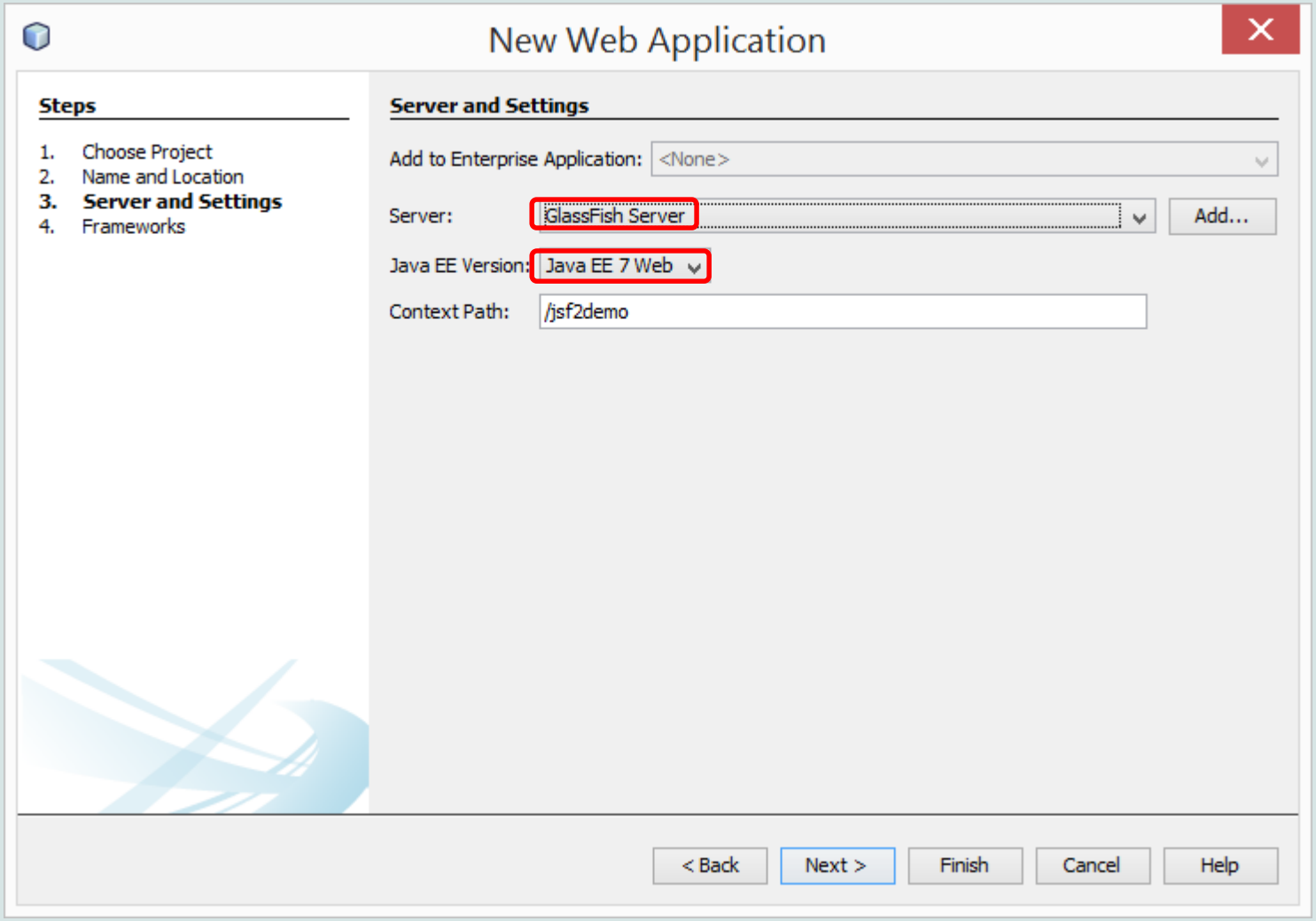
Project Folder: F:\fscj\jtech\netbeans\jsf2demo

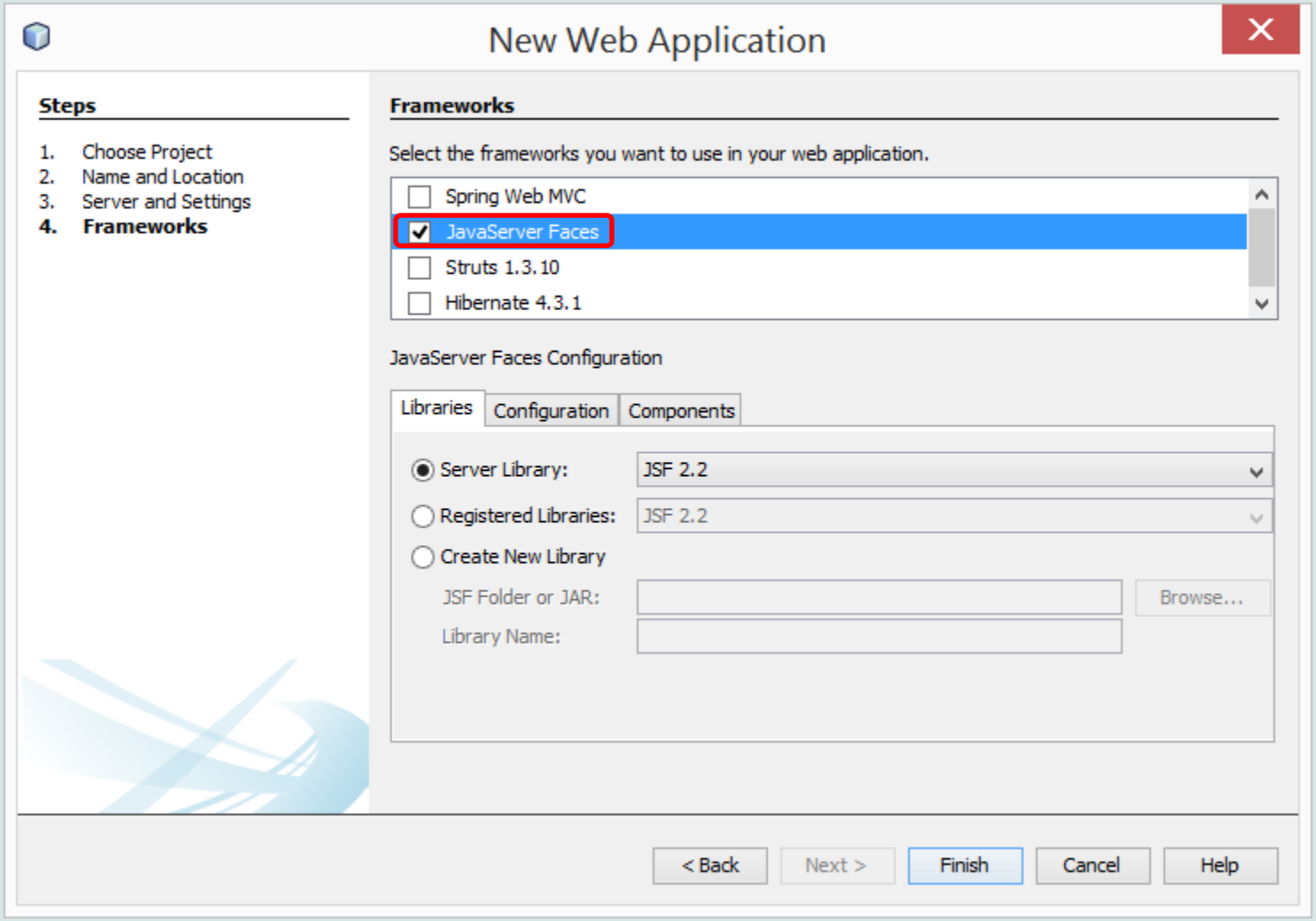
☐ Use Dedicated Folder for Storing Libraries

Libraries Folder: [Browse...]

Different users and projects can share the same compilation libraries (see Help for details).

< Back | Next > | Finish | Cancel | Help


New Web Application
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks
Server and Settings
Add to Enterprise Application: <None>
Server: GlassFish Server
Add...
Java EE Version: Java EE 7 Web
Context Path: /jsf2demo
< Back
Next >
Finish
Cancel
Help


New Web Application
Steps
1. Choose Project
2. Name and Location
3. Server and Settings
4. Frameworks
Frameworks
Select the frameworks you want to use in your web application.
Spring Web MVC
JavaServer Faces
Struts 1.3.10
Hibernate 4.3.1
JavaServer Faces Configuration
Libraries
Configuration
Components
Server Library:
JSF 2.2
Registered Libraries:
JSF 2.2
Create New Library
JSF Folder or JAR:
Browse...
Library Name:
< Back
Next >
Finish
Cancel
Help

XML Declaration (must always be first in the document if present)

XHTML Version

Tags

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Hello from Facelets
    </h:body>
</html>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Hello from Facelets
    </h:body>
</html>
```

- xmlns = XML namespace
  - similar to Java packages
  - helps avoid naming conflicts
- xmlns=http://www.w3.org/1999/xhtml
  - default standard namespace
- xmlns:h=http://xmlns.jcp.org/jsf/html
  - JSF tag library (attribute must include ':h')

- An XML document consists of elements described by tags
  - Tags are enclosed in angle brackets
    - Start tags are identified by < >
    - End tags are identified by </ >
      - (forward slash after left angle bracket)
  - An element is enclosed between a start tag and an end tag
  - XML tags are case-sensitive (HTML tags are not)
- XML elements are organized in a tree-like hierarchy, starting at the root element
  - Elements may contain subelements
  - All subelements must be enclosed inside the root element
  - The root element in XHTML is defined using the html tag

You can now display the page in index.xhtml by right-clicking on index.xhtml in the projects pane and choose *Run File*. The page is displayed in a browser, as shown in Figure 33.5.

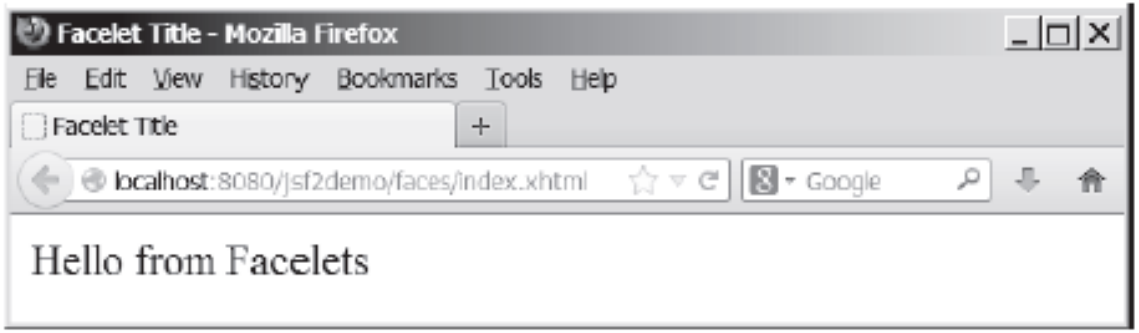
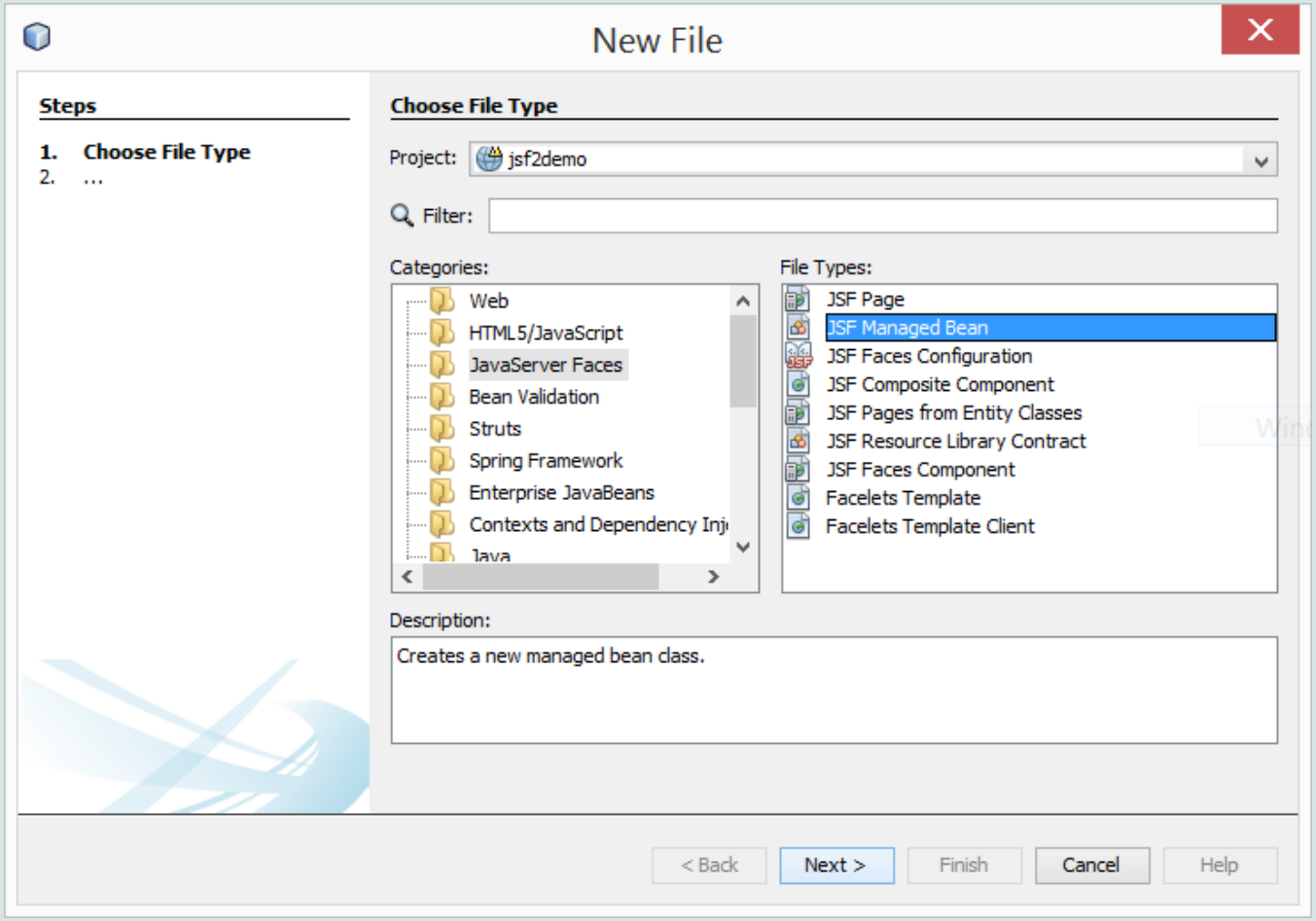


FIGURE 33.5 The index.xhtml is displayed in the browser.

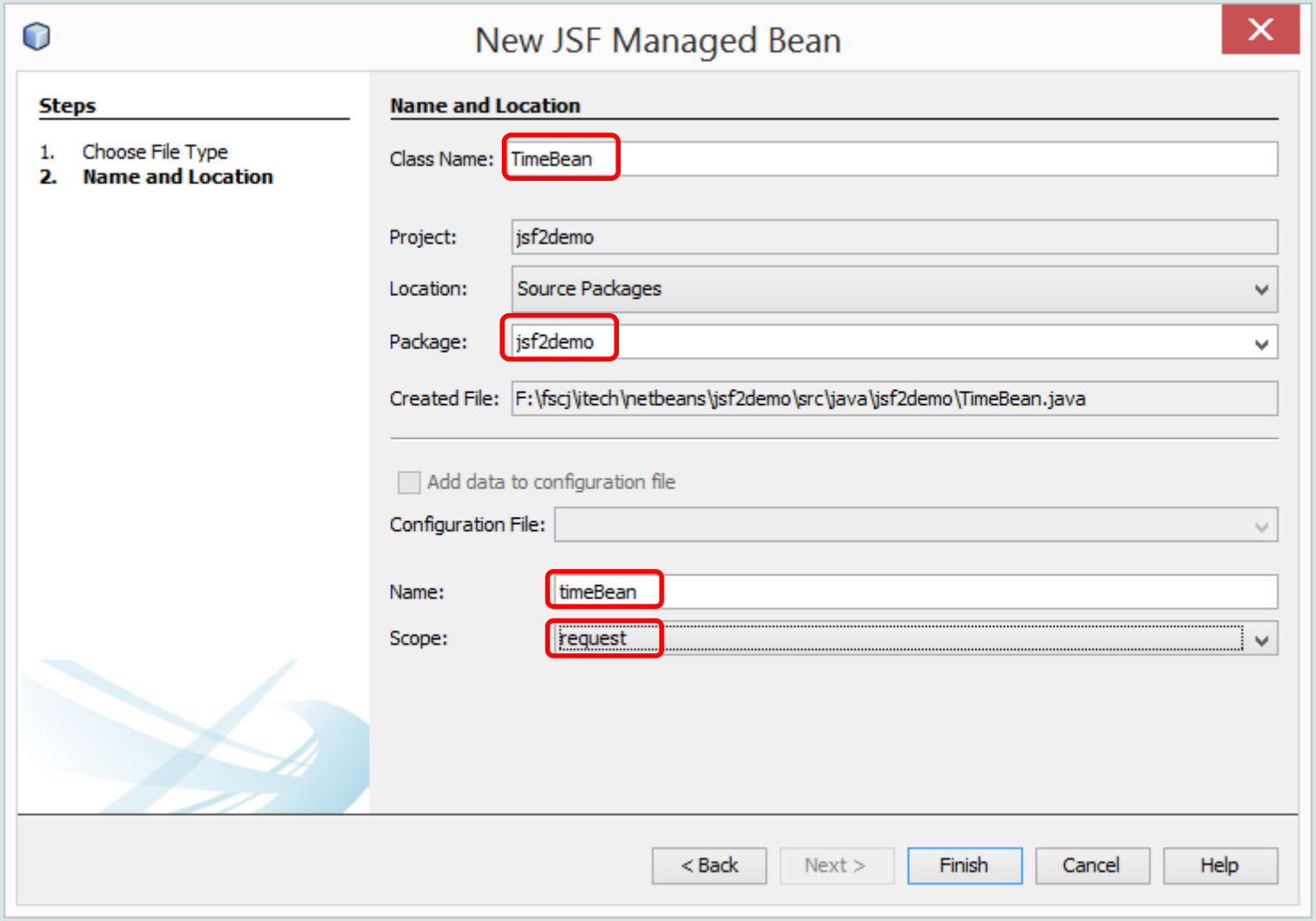
Managed JavaBeans for JSF

- JavaBean properties make them well-suited for use in JSF applications
  - Naming getters (accessors) and setters (mutators) correctly is critical: getProperty() and setProperty() must be used.
  - Properties may also be read-only (no setter provided) or write-only (no getter provided)
- Example:
  - Develop a JSF Facelet to display the current time
  - Create a JavaBean named TimeBean which includes a getTime() method that returns the current time
  - The facelet invokes this method

- Right-click on jsf2demo project node and select New/Other
  - In New File dialog select JavaServer Faces/JSF Managed Bean



- Configure the bean as shown below:



- Add the getTime() method:

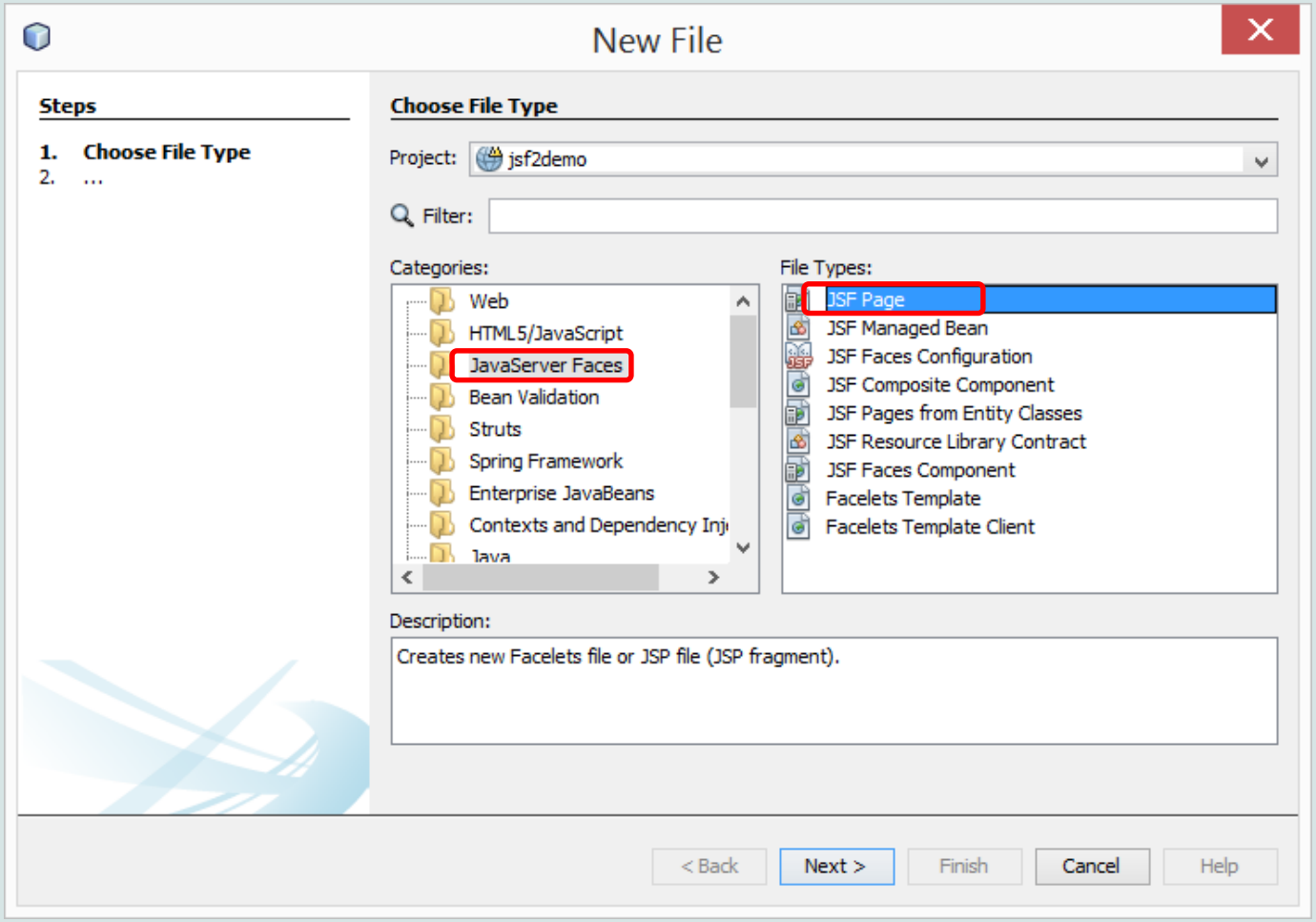
```java
package jsf2demo;

import javax.inject.Named;
import javax.enterprise.context.RequestScoped;

/**
 *
 * @author david
 */
@Named(value = "timeBean")
@RequestScoped
public class TimeBean {

    /**
     * Creates a new instance of TimeBean
     */
    public TimeBean() {
    }

    public String getTime() {
        return new java.util.Date().toString();
    }
}
```

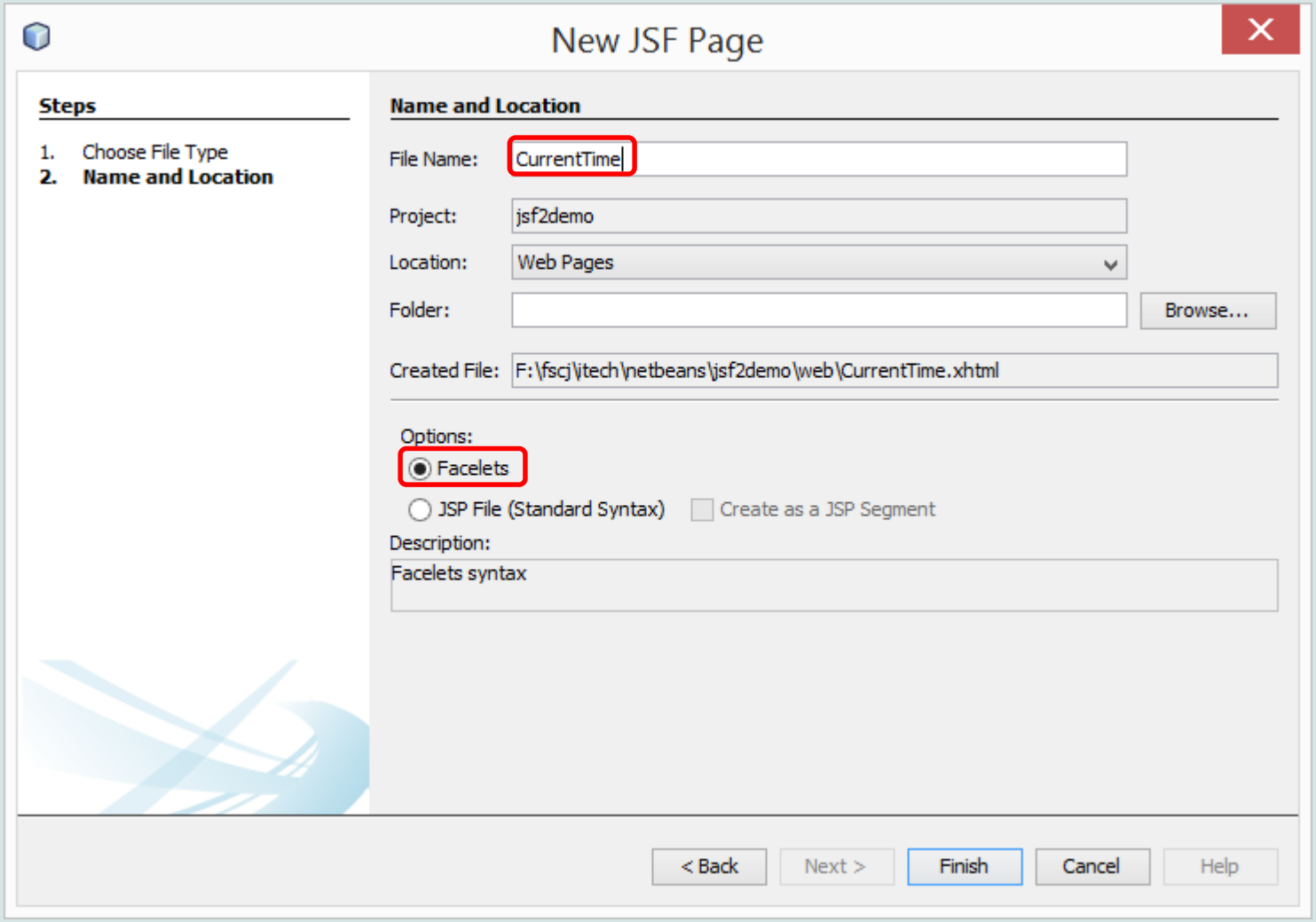
- The @Named annotation tells NetBeans to configure this bean to be used by a Facelet.
- @RequestScope specifies the scope of the bean is within a request (vs. general application availability). The bean lives long enough for a single HTTP request-response cycle.

```
@Named(value = "timeBean")
@RequestScoped
public class TimeBean {

    /**
     * Creates a new instance of TimeBean
     */
    public TimeBean() {
    }

    public String getTime() {
        return new java.util.Date().toString();
    }
}
```

JSF Expressions

- Using a JSF Expression to continue our time application
  - Right-click on the project node and select New/Other to create a JSF Page




New JSF Page
Steps
1. Choose File Type
2. Name and Location
Name and Location
File Name: CurrentTime
Project: jsf2demo
Location: Web Pages
Folder:
Browse...
Created File: F:\fscj\itech\netbeans\jsf2demo\web\CurrentTime.xhtml
Options:
Facelets
JSP File (Standard Syntax)
Create as a JSP Segment
Description:
Facelets syntax
< Back
Next >
Finish
Cancel
Help

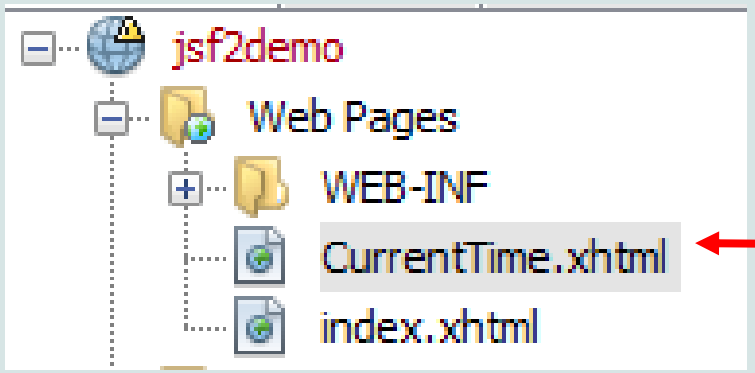
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Tr
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Hello from Facelets
    </h:body>
</html>
```

Make the changes circled in red

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transit
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Display Current Time</title>
        <meta http-equiv="refresh" content ="60" />
    </h:head>
    <h:body>
        The current time is #{timeBean.time}
    </h:body>
</html>
```

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transit
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Display Current Time</title>
        <meta http-equiv="refresh" content ="60" />
    </h:head>
    <h:body>
        The current time is #{timeBean.time}
    </h:body>
</html>
```

This is the JSF Expression which refers to our JavaBean. JSF expressions bind JavaBeans objects with Facelets. The name "timeBean" is established by the "@Named" annotation in the class we defined



Right-click on "CurrentTime.xhtml" and select "Run File" to run the application

Line 11 uses a JSF expression `#{timeBean.time}` to obtain the current time. `timeBean` is an object of the `TimeBean` class. The object name can be changed in the `@Named` annotation (line 6 in Listing 33.2) using the following syntax:

```
@Named(name = "anyObjectName")
```

By default, the object name is the class name with the first letter in lowercase.

JSF GUI Components

Table 33.1 lists some of the commonly used elements. The tags with the **h** prefix are in the JSF HTML Tag library. The tags with the **f** prefix are in the JSF Core Tag library.

TABLE 33.1 JSF GUI Form Elements

JSF Tag	*Description*
`h:form`	inserts an XHTML form into a page.
`h:panelGroup`	similar to a JavaFX FlowPane.
`h:panelGrid`	similar to a JavaFX GridPane.
`h:inputText`	displays a textbox for entering input.
`h:outputText`	displays a textbox for displaying output.
`h:inputTextArea`	displays a textarea for entering input.
`h:inputSecret`	displays a textbox for entering password.
`h:outputLabel`	displays a label.
`h:outputLink`	displays a hypertext link.
`h:selectOneMenu`	displays a combo box for selecting one item.
`h:selectOneRadio`	displays a set of radio button.
`h:selectManyCheckbox`	displays checkboxes.
`h:selectOneListbox`	displays a list for selecting one item.
`h:selectManyListbox`	displays a list for selecting multiple items.
`f:selectItem`	specifies an item in an h:selectOneMenu, h:selectOneRadio, or h:selectManyListbox.
`h:message`	displays a message for validating input.
`h:dataTable`	displays a data table.
`h:column`	specifies a column in a data table.
`h:graphicImage`	displays an image.

Creating a Registration Form

LISTING 33.4 StudentRegistrationForm.xhtml

```
<?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"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
  <h:head>
    <title>Student Registration Form</title>
  </h:head>
  <h:body>
    <h:form>
      <!-- Use h:graphicImage -->
      <h3>Student Registration Form
        <h:graphicImage name="usIcon.gif" library="image"/>
      </h3>

      <!-- Use h:panelGrid -->
      <h:panelGrid columns="6" style="color:green">
        <h:outputLabel value="Last Name"/>
        <h:inputText id="lastNameInputText" />
        <h:outputLabel value="First Name" />
        <h:inputText id="firstNameInputText" />
        <h:outputLabel value="MI" />
        <h:inputText id="miInputText" size="1"   />
      </h:panelGrid>

      <!-- Use radio buttons -->
      <h:panelGrid columns="2">
        <h:outputLabel>Gender  </h:outputLabel>
```

jsf core namespace

graphicImage

h:panelGrid

h:outputLabel

h:inputText

```
30          <h:selectOneRadio id="genderSelectOneRadio">
31            <f:selectItem itemValue="Male"
32                          itemLabel="Male"/>
33            <f:selectItem itemValue="Female"
34                          itemLabel="Female"/>
35          </h:selectOneRadio>
36        </h:panelGrid>
37
38        <!-- Use combo box and list -->
39        <h:panelGrid columns="4">
40          <h:outputLabel value="Major  "/>
41          <h:selectOneMenu id="majorSelectOneMenu">
42            <f:selectItem itemValue="Computer Science"/>
43            <f:selectItem itemValue="Mathematics"/>
44          </h:selectOneMenu>
45          <h:outputLabel value="Minor  "/>
46          <h:selectManyListbox id="minorSelectManyListbox">
47            <f:selectItem itemValue="Computer Science"/>
48            <f:selectItem itemValue="Mathematics"/>
49            <f:selectItem itemValue="English"/>
50          </h:selectManyListbox>
51        </h:panelGrid>
52
```

```
53        <!-- Use check boxes -->
54        <h:panelGrid columns="4">
55          <h:outputLabel value="Hobby: "/>
56          <h:selectManyCheckbox id="hobbySelectManyCheckbox">
57            <f:selectItem itemValue="Tennis"/>
58            <f:selectItem itemValue="Golf"/>
59            <f:selectItem itemValue="Ping Pong"/>
60          </h:selectManyCheckbox>
61        </h:panelGrid>
62
63        <!-- Use text area -->
64        <h:panelGrid columns="1">
65          <h:outputLabel>Remarks:</h:outputLabel>
66          <h:inputTextarea id="remarksInputTextarea"
67                           style="width:400px; height:50px;" />
68        </h:panelGrid>
69
70        <!-- Use command button -->
71        <h:commandButton value="Register" />
72      </h:form>
73    </h:body>
74  </html>
```

Processing the Form

- We need to bind each input element with a property in a managed bean ("managed" means the bean is registered with JSF)

```
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

@Named(value = "registration")
@RequestScoped
public class RegistrationJSFBean {
    private String lastName;
    private String firstName;
    private String mi;
    private String gender;
    private String major;
    private String[] minor;
    private String[] hobby;
    private String remarks;

    public RegistrationJSFBean() {
    }
```

Running the Application

Last Name Smith First Name John MI

Gender Male Female

Major Mathematics Minor Computer Science Mathematics English

Hobby: Tennis Golf Ping Pong

Remarks:

Register

You entered
Last Name: Smith
First Name: John
MI:
Gender: Male
Major: Mathematics
Minor: Mathematics
Hobby: Tennis
Remarks:

Session Tracking

- Scope refers to the lifetime of a bean
  - A request-scoped bean is alive in a single HTTP request
    - After the request is processed, the bean is no longer alive
  - A view-scoped bean lives as long as you are on the same JSF page
  - A session-scoped bean is alive for the entire Web session between a client and the server
  - An application-scoped bean lives as long as the Web application runs

LISTING 33.9 GuessNumberJSFBean.java

```
1  package jsf2demo;
2
3  import javax.inject.Named;
4  import javax.faces.view.ViewScoped;
5
6  @Named(value = "guessNumber")
7  @ViewScoped
8  public class GuessNumberJSFBean {
9    private int number;
10   private String guessString;
```

- This bean uses the view scope; the bean remains alive as long as the view is not changed
  - The bean is created when the page is displayed for the first time
  - A random number between 0 and 99 is assigned to number (line 13) when the bean is created. This number will not change as long as the bean is alive in the same view
- What happens if the scope is changed to the request scope?
  - Every time the page is refreshed, JSF creates a new bean with a new random number
- What happens if the scope is changed to the **session** scope?
  - The bean will be alive as long as the browser is alive
- What happens if the scope is changed to the **application** scope?
  - The bean will be created once when the application is launched from the server, so every client will use the same random number

Validating Input

- JSF provides input validators

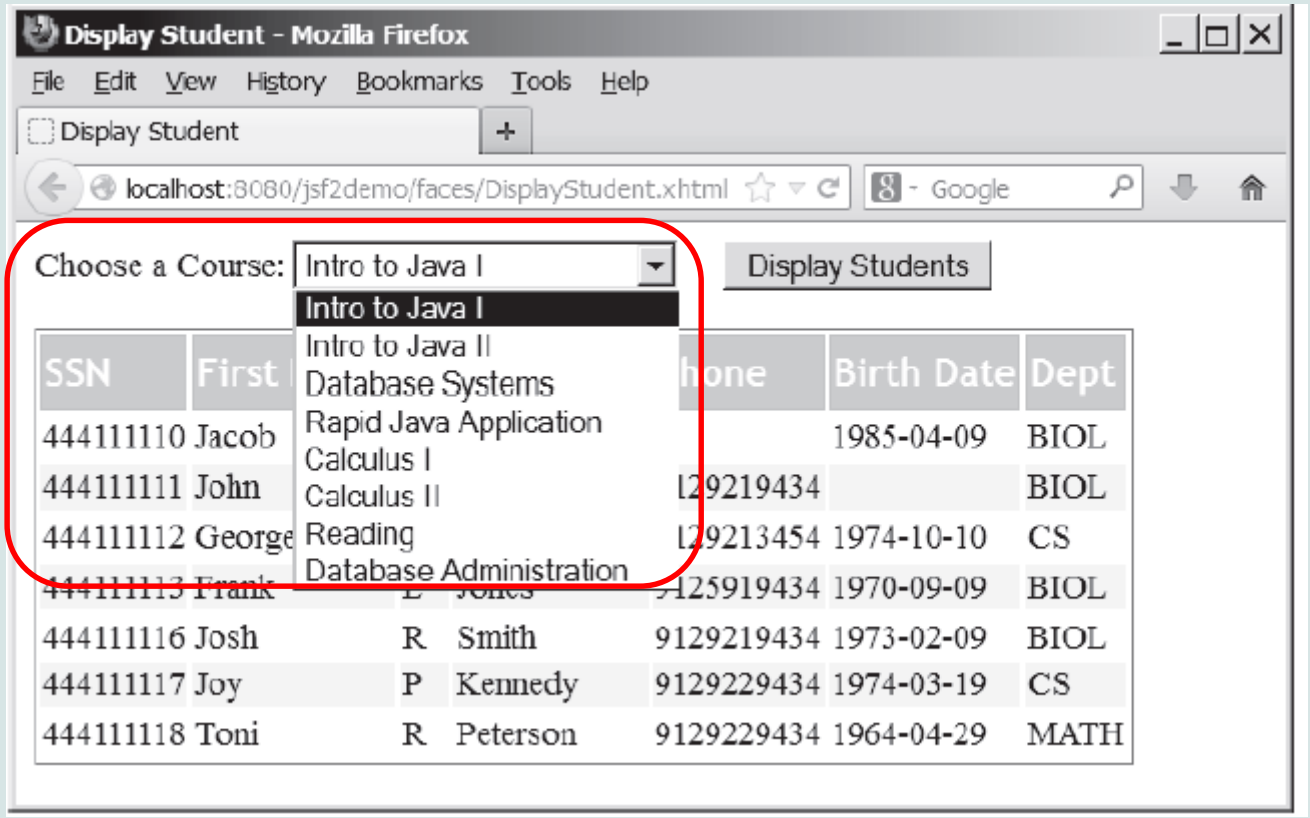
TABLE 33.2 JSF Input Validator Tags

JSF Tag	*Description*
`f:validateLength`	validates the length of the input.
`f:validateDoubleRange`	validates whether numeric input falls within acceptable range of double values.
`f:validateLongRange`	validates whether numeric input falls within acceptable range of long values.
`f:validateRequired`	validates whether a field is not empty.
`f:validateRegex`	validates whether the input matches a regualar expression.
`f:validateBean`	invokes a custom method in a bean to perform custom validation.

```
<h:outputLabel value="Age:" />
<h:inputText id="ageInputText" required="true"
  requiredMessage="Age is required"
  validatorMessage="Age must be between 16 and 120"
  value="#{validateForm.ageString}">
  <f:validateLongRange minimum="16" maximum="120"/>
</h:inputText>
<h:message for="ageInputText" style="color:red"/>
```

Binding to Databases with Facelets

- Bind the combo box with the Course table in our database



LISTING 33.13 CourseNameJSFBean.java

```
package jsf2demo;

import java.sql.*;
import java.util.ArrayList;
import javax.enterprise.context.ApplicationScoped;
import javax.inject.Named;

@Named(value = "courseName")
@ApplicationScoped
public class CourseNameJSFBean {
  private PreparedStatement studentStatement = null;
  private String choice; // Selected course
  private String[] titles; // Course titles

  /** Creates a new instance of CourseName */
  public CourseNameJSFBean() {
    initializeJdbc();
  }

  /** Initialize database connection */
  private void initializeJdbc() {
    try {
      Class.forName("com.mysql.jdbc.Driver");
      System.out.println("Driver loaded");

      // Connect to the sample database
      Connection connection = DriverManager.getConnection(
        "jdbc:mysql://localhost/javabook", "scott", "tiger");

      // Get course titles
      PreparedStatement statement = connection.prepareStatement(
        "select title from course");

      ResultSet resultSet = statement.executeQuery();
```

```java

      // Store resultSet into array titles
      ArrayList<String> list = new ArrayList<>();
      while (resultSet.next()) {
        list.add(resultSet.getString(1));
      }
      titles = new String[list.size()]; // Array for titles
      list.toArray(titles); // Copy strings from list to array

      // Define a SQL statement for getting students
      studentStatement = connection.prepareStatement(
        "select Student.ssn, "
        + "student.firstName, Student.mi, Student.lastName, "
        + "Student.phone, Student.birthDate, Student.street, "
        + "Student.zipCode, Student.deptId "
        + "from Student, Enrollment, Course "
        + "where Course.title = ? "
        + "and Student.ssn = Enrollment.ssn "
        + "and Enrollment.courseId = Course.courseId;");
    }
    catch (Exception ex) {
      ex.printStackTrace();
    }
  }
```

```java

    public String[] getTitles() {
      return titles;
    }

    public String getChoice() {
      return choice;
    }

    public void setChoice(String choice) {
      this.choice = choice;
    }

    public ResultSet getStudents() throws SQLException {
      if (choice == null) {
        if (titles.length == 0)
          return null;
        else
          studentStatement.setString(1, titles[0]);
      }
      else {
        studentStatement.setString(1, choice); // Set course title
      }

      // Get students for the specified course
      return studentStatement.executeQuery();
    }
  }
```