Student ID__________________Student Name____________________________________

Web Applications Architecture and Frameworks DE

Final Exam June 25, 2016

PRIVATE AND CONFIDENTIAL

1. **Allotted exam duration is 2 hours.**
2. **Closed book/notes.**
3. **No personal items including electronic devices (cell phones, computers, calculators, PDAs).**
4. **Cell phones must be turned in to your proctor before beginning exam.**
5. **No additional papers are allowed. Sufficient blank paper is included in the exam packet.**
6. **Exams are copyrighted and may not be copied or transferred.**
7. **Restroom and other personal breaks are not permitted.**
8. **Total exam including questions and scratch paper must be returned to the proctor.**

2 blank pages are provided for writing the solutions and/or scratch paper. All 2 pages must be handed in with the exam

BE VERY CAREFUL WITH THE GIVEN 2 HOURS AND USE YOUR TIME WISELY. THE ALLOTED TIME IS GIVEN FOR EVERY QUESTION.

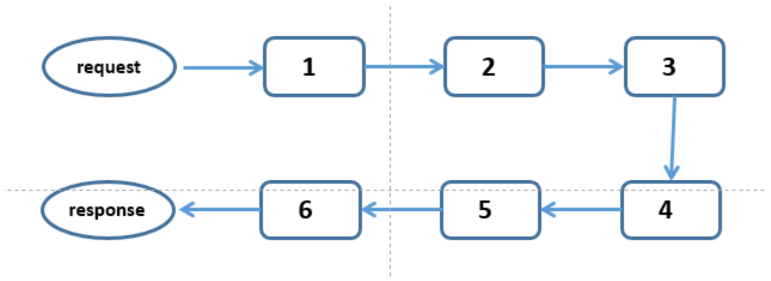
Write your **name and student id** at the top of this page.

All your answers have to be written in the designated areas on the exam paper. If you make a mistake, you can use the extra blank pages.

Write clearly. If I cannot read it, I cannot give any points.

Question 1: [10 points] {10 minutes}

The JSF lifecycle consists of 6 phases:



a. Give the name of every phase and in one sentence describe what JSF does in this phase.

Phase name	**Short description of what JSF does in this phase**
1. Restore view	**Create/retrieve component tree**
2. Apply request	**The values of the components in the component tree are updated with the values from the request**
3. Process validations	**Validate the values in the component tree**
4. Update model values	**Find the backingbean and update the values in the backing bean**
5. Invoke application	**Call the registered action listeners**
6. Render response	**Create the Http response**

b. JSF splits the JSF lifecycle into 2 parts: execute and render

Give the number(s) of the phases that belong to the execute part

1, 2, 3, 4, 5

Give the number(s) of the phases that belong to the render part

6

Question 2: [10 points] {10 minutes}

Given are 2 code snippets:

Snippet 1:
```
<h:form
<h:selectOneMenu value="#{bean.value}" valueChangeListener="#{bean.changeListener}">
   <f:selectItems ... />
</h:selectOneMenu>
<h:/form>
```

Snippet 2:
```
<h:selectOneMenu value="#{bean.value}">
   <f:selectItems ... />
   <f:ajax listener="#{bean.ajaxListener}" />
</h:selectOneMenu>
```

Explain clearly the difference in behavior that you will observe when you use snippet 1 or snippet 2.

Snippet 1:

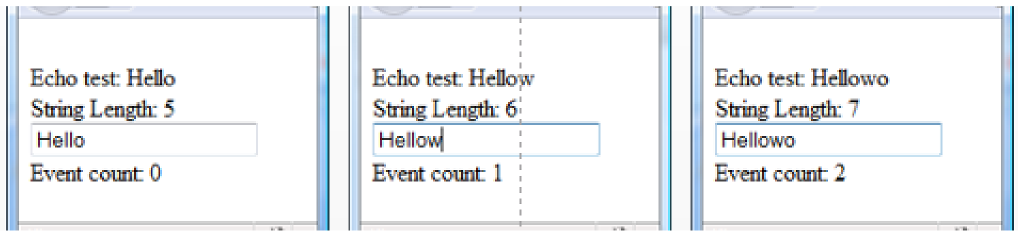
The code of Snippet 1 will not send a http request to the server if you select an item
You need onChange=”submit()” if you want to send a http request to the server
It will send a synchronous request to the server
The whole page will be refreshed.

Snippet 2:

The code of Snippet 2 will send a http request to the server if you select an item
It is an asynchronous event
There will only a partial page update if we add a render attribute to the ajax tag

Question 3: [15 points] {20 minutes}

Suppose we want the following behavior of a JSF application:



```
Echo test: Hello
String Length: 5
Hello
Event count: 0
```

```
Echo test: Hellow
String Length: 6
Hellow
Event count: 1
```

```
Echo test: Hellowo
String Length: 7
Hellowo
Event count: 2
```

Whenever you type a character in the input field, the page shows the following info:

- The first line just shows the same string as is shown in the input field
- The second line shows the length of the string as is shown in the input field
- The bottom line shows how many times a new character is typed in the input field

For this exercise you have to use AJAX. Modify the given code (on the next page) so that we get the given behavior

```
<h:form >
        <br/>
        Echo test: <h:outputText id="out" value="#{listenBean.hello}"/>
        <br/>
        String Length: <h:outputText id="count" value="#{listenBean.length}"/>
        <br/>
        <h:inputText id="in" value="#{listenBean.hello}" >
          <f:ajax event="keyup" render="out count eventcount"
                listener="#{listenBean.update}" />
         </h:inputText>
        <br/>
        Event count: <h:outputText id="eventcount" value="#{listenBean.eventCount}"/>
      </h:form>
```

```
@ManagedBean
@ViewScoped
public class ListenBean implements Serializable{
  private String hello = "Hello";
  private int length = hello.length();
  private int eventCount = 0;

  public void update(AjaxBehaviorEvent event) {
      length = hello.length();
      eventCount++;
   }

  …
}
```

Question 4: [15 points] {20 minutes}

Suppose we have a JSF form with an inputText like this
<h:inputText *value="#{studentForm.student}"* >

The form expects that students are entered in the format ***"name-age"***. Examples are ***John-66*** or ***Frank-22***

The student in the managedBean is of type Student:

```
@ManagedBean
@RequestScoped
public class StudentForm {
   private Student student;
... //constructor, getters and setters are not shown
}

public class Student {
  private String name;
  private Integer age;
... //constructor, getters and setters are not shown
}
```

Write a JSF custom converter that converts the string that is typed in the inputText to a Student class, and also the other way around (from Student class to String)

```
@FacesConverter( value="studentConverter" )
public class Studentconverter  implements Converter {
 public Object getAsObject(FacesContext context, UIComponent component, String value) {
   if ("".equals(value)){
     return null;
   }
   Student student= new Student();
   String[] parts = value.split("-");
   String student = parts[0];
   String age = parts[1];
   student.setName(name);
   student.setage(age);
   return student
 }
 public String getAsString(FacesContext context, UIComponent component, Object value) {
   Student student = (Student) value;
   return student.getName()+"-"+student.getAge();
 }
}
```

Question 5: [10 points] {10 minutes}

JSF supports different ways to apply validation to your application. Give 3 different validation techniques that are supported by JSF

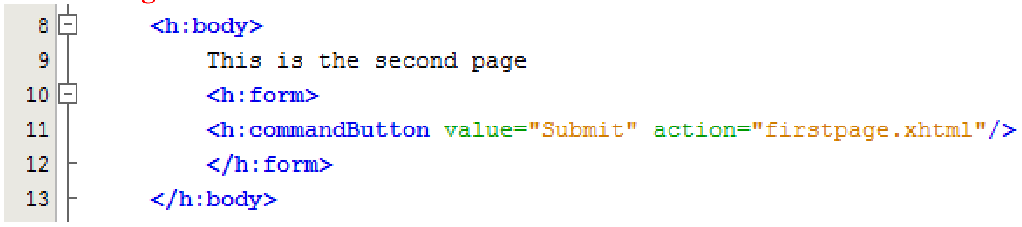
1. Build-in validators
2. Validation method in the managedbean
3. Custum validators

Bean validation

Question 6: [10 points] {15 minutes}

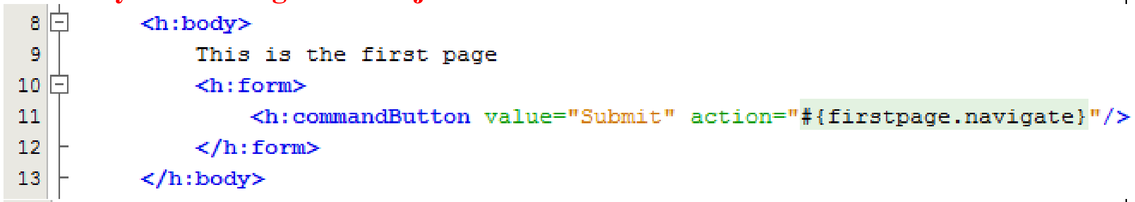
a. Give 3 different ways of how we can implement navigation in JSF. Give an example of each.

1. Static navigation in xhtml:

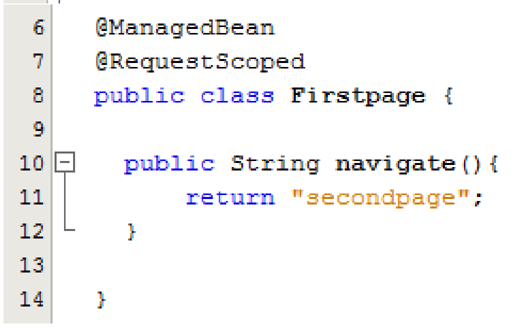


```
    <h:body>
        This is the second page
        <h:form>
        <h:commandButton value="Submit" action="firstpage.xhtml"/>
        </h:form>
    </h:body>
```

2. Dynamic navigation in a java method



```
    <h:body>
        This is the first page
        <h:form>
            <h:commandButton value="Submit" action="#{firstpage.navigate}"/>
        </h:form>
    </h:body>
```

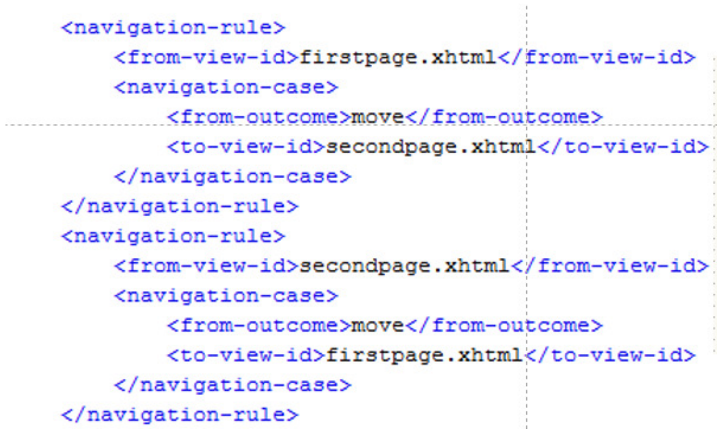


```
@ManagedBean
@RequestScoped
public class Firstpage {

  public String navigate(){
      return "secondpage";
  }

}
```

3. Configure navigation in faces-config.xml



```
<navigation-rule>
    <from-view-id>firstpage.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>move</from-outcome>
        <to-view-id>secondpage.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
<navigation-rule>
    <from-view-id>secondpage.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>move</from-outcome>
        <to-view-id>firstpage.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

b. What is the default navigation implementation of JSF, **forward** navigation or **redirect**?

forward

c. Give an example of how we can change the default navigation.

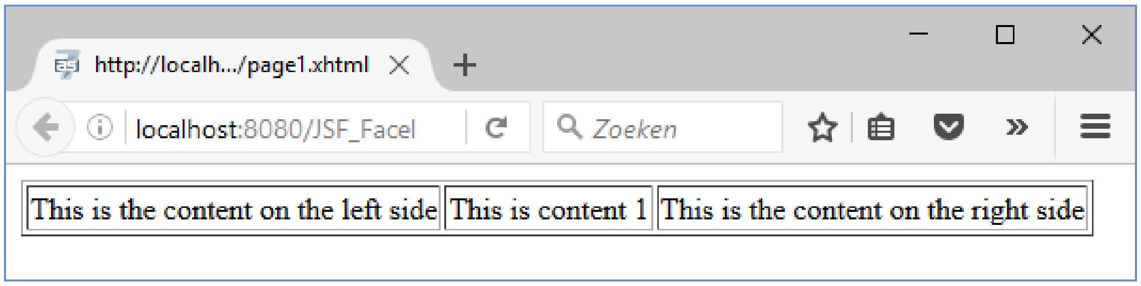
```
<h:form>
    <h:commandButton value="Submit"
                     action="firstpage?faces-redirect=true"/>
</h:form>
```

```
public String navigate(){
    return "firstpage?faces-redirect=true";
}
```

```
<navigation-rule>
    <from-view-id>firstpage.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>move</from-outcome>
        <to-view-id>secondpage.xhtml</to-view-id>
        <redirect/>
    </navigation-case>
</navigation-rule>
```

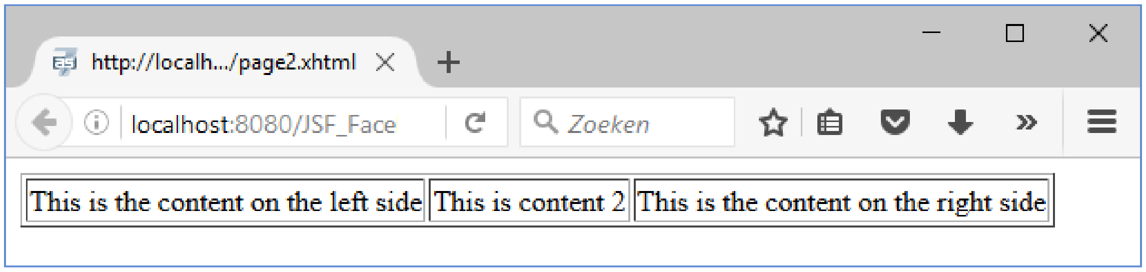
Question 7: [15 points] {15 minutes}

Write the following application with JSF and **facelets**:
page1.xhtml



This is the content on the left side	This is content 1	This is the content on the right side

page2.xhtml



This is the content on the left side	This is content 2	This is the content on the right side

Every page we have to make is divided into 3 parts: a left part, a middle part and a right part.
We need to use **facelets** to make it easy to give every page this same structure.
Given are the following files:

left.xhtml:
```
<ui:composition>
  <h:outputText value="This is the content on the left side"/>
</ui:composition>
```

right.xhtml:
```
<ui:composition>
  <h:outputText value="This is the content on the right side"/>
</ui:composition>
```

content1.xhtml:
```
<ui:composition>
      This is content 1
</ui:composition>
```

content2.xhtml:
```
<ui:composition>
      This is content 2
</ui:composition>
```

Write all the necessary files that are missing.

page1.xhtml:

```
<ui:composition template="template.xhtml">
        <ui:define name="middle">
                <ui:include src="content.xhtml" />
        </ui:define>
</ui:composition>
```

page2.xhtml:

```
<ui:composition template="template.xhtml">
        <ui:define name="middle">
                <ui:include src="content2.xhtml" />
        </ui:define>
</ui:composition>
```

template.xhtml:

```
<h:panelGrid columns="3" border="1">
        <ui:include  src="left.xhtml" />
        <ui:insert name="middle" />
        <ui:include src="right.xhtml" />
</h:panelGrid>
```

Your implementation should follow the following requirements:

1. You are **not** allowed to use HTML, JSP's or servlets.
2. You need to use **facelets**
3. **For web pages, you only need to write the code between the <body> and </body> tags. Do not write the code for namespaces**

Question 8: [10 points] {10 minutes}

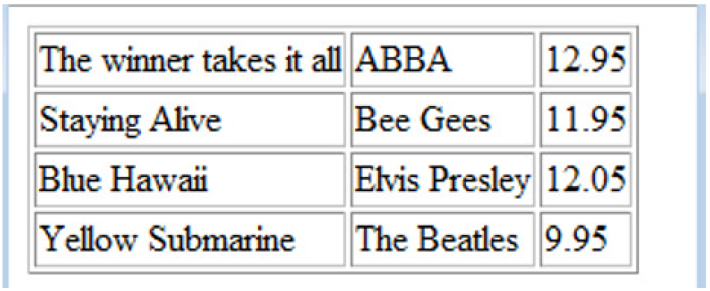
Suppose we have the following CD class:

```
public class CD {
  private String title = "";
  private String artist = "";
  private double price = 0;
  …
}
```

And the following list of CD's in a managed bean:

```
public class TheManagedbean
    private Collection<CD> cdList=new ArrayList<CD>();
```

And we want to show the content of the cdList in a JSF table like this:



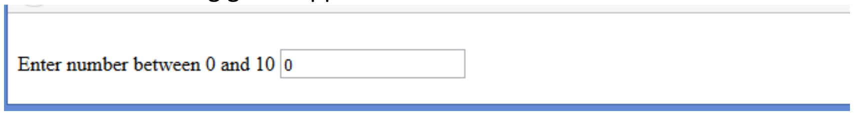
The winner takes it all	ABBA	12.95
Staying Alive	Bee Gees	11.95
Blue Hawaii	Elvis Presley	12.05
Yellow Submarine	The Beatles	9.95

Write the part of code we need in the xhtml page that shows the content of the cdList like the example given above:

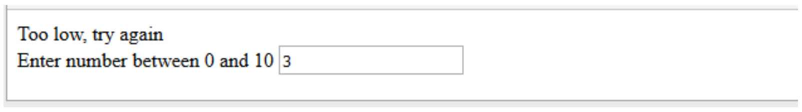
```
<h:dataTable value="#{dataTablebackingbean.cdList}" var="mycd" border="1">
        <h:column>
          <h:outputText value="#{mycd.title}"/>
        </h:column>
        <h:column>
          <h:outputText value="#{mycd.artist}"/>
        </h:column>
        <h:column>
          <h:outputText value="#{mycd.price}"/>
        </h:column>
</h:dataTable>
```

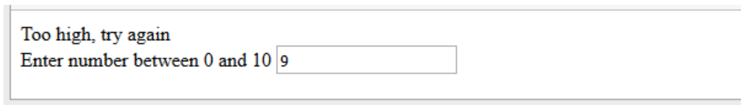
Question 9: [15 points] {20 minutes}

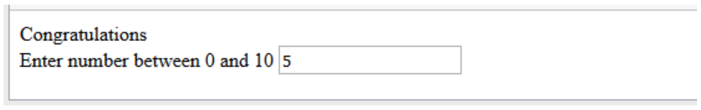
Write the following guess application with JSF:


Enter number between 0 and 10 0

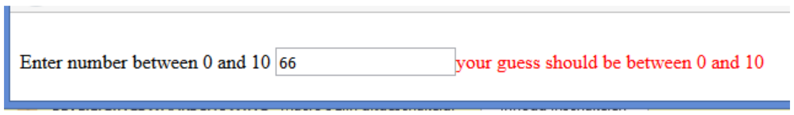
The applications ask to enter a number between 0 and 10. The number that needs to be guessed is hard coded at the number 5.
When you type a number in the input field, the application will show if this number is too high, too low or correct:

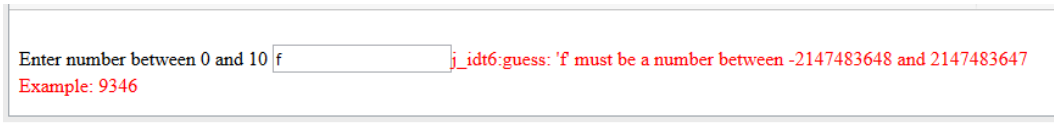

Too low, try again
Enter number between 0 and 10 3


Too high, try again
Enter number between 0 and 10 9


Congratulations
Enter number between 0 and 10 5

When you enter an incorrect input, the application will show this:


Enter number between 0 and 10 66 your guess should be between 0 and 10


Enter number between 0 and 10 f j_idt6:guess: 'f' must be a number between -2147483648 and 2147483647
Example: 9346

Add the missing code on the next page.
Your implementation should follow the following requirements:

1. You are **not** allowed to use HTML, JSP's or servlets.
2. Do **not** write getter and setter methods

For web pages, you only need to write the code between the <body> and </body> tags. Do not write the code for namespaces

```
  <h:body>
    <h:outputText id="result" value="#{guessBean.result}"/>
    <br/>
     <h:form>
        Enter number between 0 and 10
        <h:inputText id="guess" value="#{guessBean.guess}" validatorMessage="your guess should be
between 0 and 10">
            <f:ajax event="keyup" render="result errormessage" listener="#{guessBean.checkguess}"/>
            <f:validateLongRange maximum="10" minimum="0"/>
        </h:inputText>
        <h:message id="errormessage" for="guess" style="color: red" />
     </h:form>
  </h:body>


@ManagedBean
@RequestScoped
public class GuessBean {

  private int guess, toBeGuessed;
  private String result="";

  public GuessBean() {
     toBeGuessed=5;
  }

  public void checkguess(AjaxBehaviorEvent event) {
     System.out.println("yes");
     if (guess == toBeGuessed){
        result = "Congratulations";
        return;
     }
     if (guess < toBeGuessed){
        result = "Too low, try again";
     }
     else{
        result = "Too high, try again";
     }
  }
```