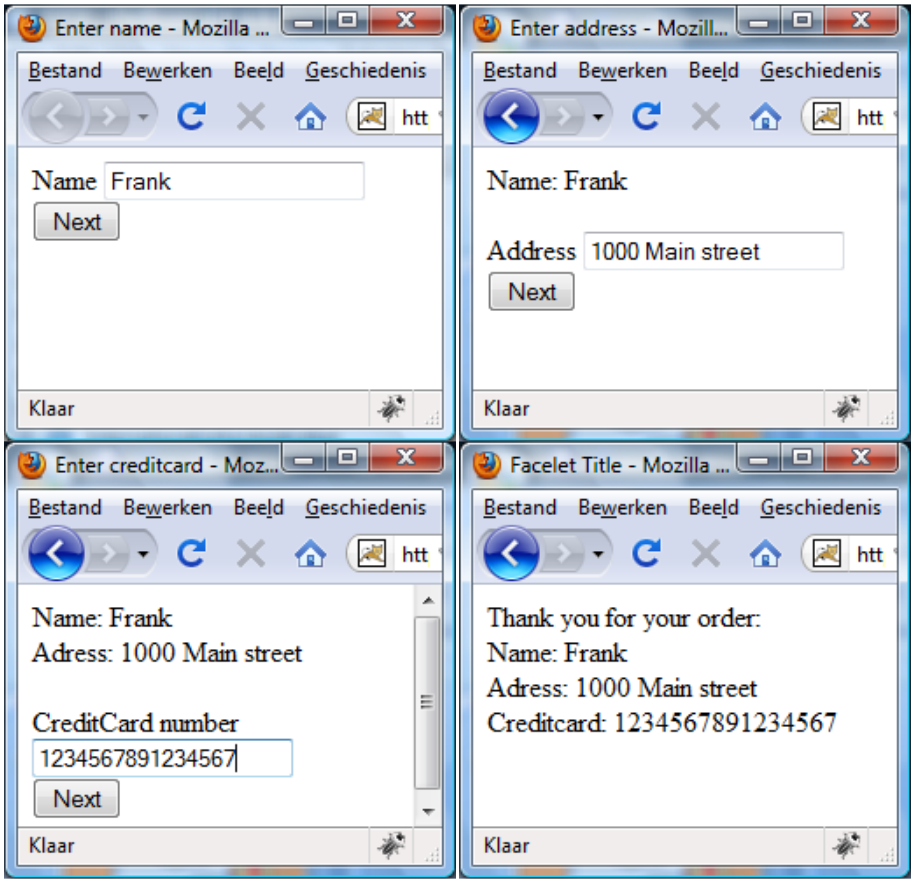
Practice exam

Question 1 OrderApplication [30 points] {35 minutes}

Write a JSF application with the following behavior:



The application consists of 4 pages, where you fill an order class with the attributes name, address and creditcard. You start with the first page, you fill in the name, and you click the **Next** button. The second page is shown with the name you entered, and you fill in your address, etc.

Complete the partial given code such that the application works with the given behavior. You have to implement all java code, required annotations and JSF tags that are missing.
IMPORTANT: do NOT write getter and setter methods!

name.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
     <title>Enter name</title>
  </h:head>
  <h:body>


     <h:form>
        Name <h:inputText value="#{namebean.order.name}"/>
        <br />
        <h:commandButton value="Next" action="address.xhtml"/>
     </h:form>




  </h:body>
</html>
```

address.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
     <title>Enter address</title>
  </h:head>
  <h:body>



    <h:form>
       Name: <h:outputText value="#{addressbean.order.name}"/>
       <br />
       <br />
        Address <h:inputText value="#{addressbean.order.address}"/>
        <br />
        <h:commandButton value="Next" action="creditcard.xhtml"/>
     </h:form>

  </h:body>
</html>
```

creditcard.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
     <title>Enter creditcard</title>
  </h:head>
  <h:body>


    <h:form>
       Name: <h:outputText value="#{creditcardbean.order.name}"/><br />
       Adress: <h:outputText value="#{creditcardbean.order.address}"/>
       <br />
       <br />
        CreditCard number <h:inputText value="#{creditcardbean.order.creditcard}"/>
        <br />
        <h:commandButton value="Next" action="confirmation.xhtml"/>
     </h:form>


  </h:body>
</html>
```

confirmation.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
  <h:head>
    <title>Facelet Title</title>
  </h:head>
  <h:body>


      Thank you for your order:
      <br/>
      Name: <h:outputText value="#{confirmationbean.order.name}"/><br />
      Adress: <h:outputText value="#{confirmationbean.order.address}"/><br />
      Creditcard: <h:outputText value="#{confirmationbean.order.creditcard}"/><br />


  </h:body>
</html>
```

NameBean.java

```
@ManagedBean
@RequestScoped
public class Namebean {

   @ManagedProperty(value="#{order}")
   private Order order;

 //getters and setters are not shown
}
```

AddressBean.java

```
@ManagedBean
@RequestScoped
public class Addressbean {
   @ManagedProperty(value="#{order}")
   private Order order;

 //getters and setters are not shown
}
```

```
@ManagedBean
@RequestScoped
public class Creditcardbean {
   @ManagedProperty(value="#{order}")
   private Order order;

  //getters and setters are not shown
}
```

Confirmationbean.java

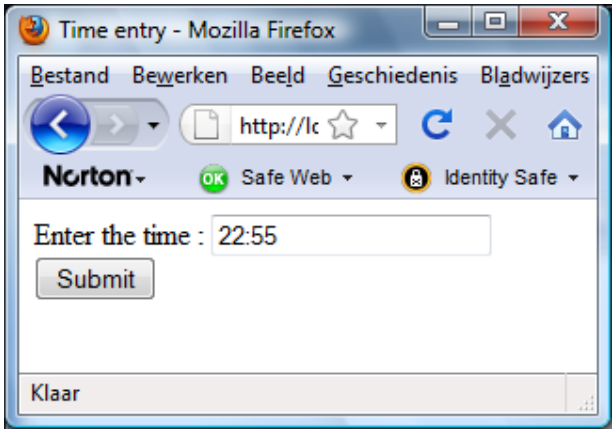
```
@ManagedBean
@RequestScoped
public class Confirmationbean {
   @ManagedProperty(value="#{order}")
   private Order order;

  //getters and setters are not shown
}
```

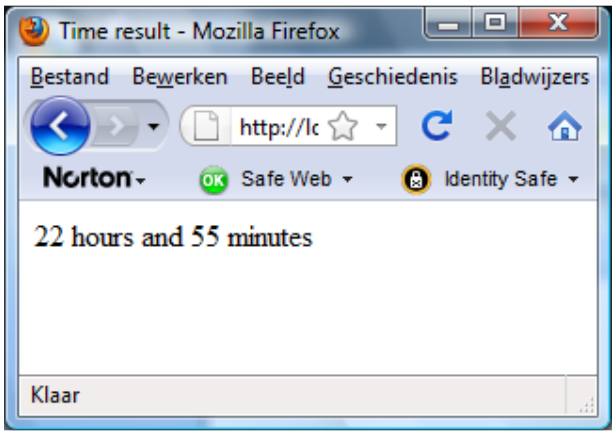
Order.java

```
@ManagedBean
@SessionScoped
public class Order {

   private String name;
   private String address;
   private String creditcard;

   //getters and setters are not shown
}
```

Question 2 TimeApplication [20 points] {20 minutes}

Write a JSF application with the following behavior:





The application consists of 2 pages, an entry page and a result page.
In the entry page you type in a string of the format **hh:mm** where hh are the hours and mm are the minutes. For example when you enter 22:55 and click the Submit button, the result page is shown with the text 22 hours and 55 minutes. And when you enter 12:15 and click the Submit button, the result page is shown with the text 12 hours and 15 minutes.
Write your own converter to implement this behavior.

Complete the partial given code such that the application works with the given behavior. You have to implement ALL java code, required annotations and JSF tags that are missing.
IMPORTANT: do not write getter and setter methods!

entry.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>Time entry</title>
  </h:head>
  <h:body>
    <h:form>
      Enter the time :


      <h:inputText value="#{timeBean.time}">
        <f:converter converterId="timeConverter"/>
      </h:inputText>
      <br />
      <h:commandButton value="Submit"
                action="result.xhtml"/>


    </h:form>
  </h:body>
</html>
```

result.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title>Time result</title>
  </h:head>
  <h:body>


    <h:outputText value="#{timeBean.time}">
      <f:converter converterId="timeConverter"/>
    </h:outputText>


  </h:body>
</html>
```

TimeBean.java

```
@ManagedBean
@RequestScoped
public class TimeBean {

   MyTime time;

   //getters and setters are not shown
}
```

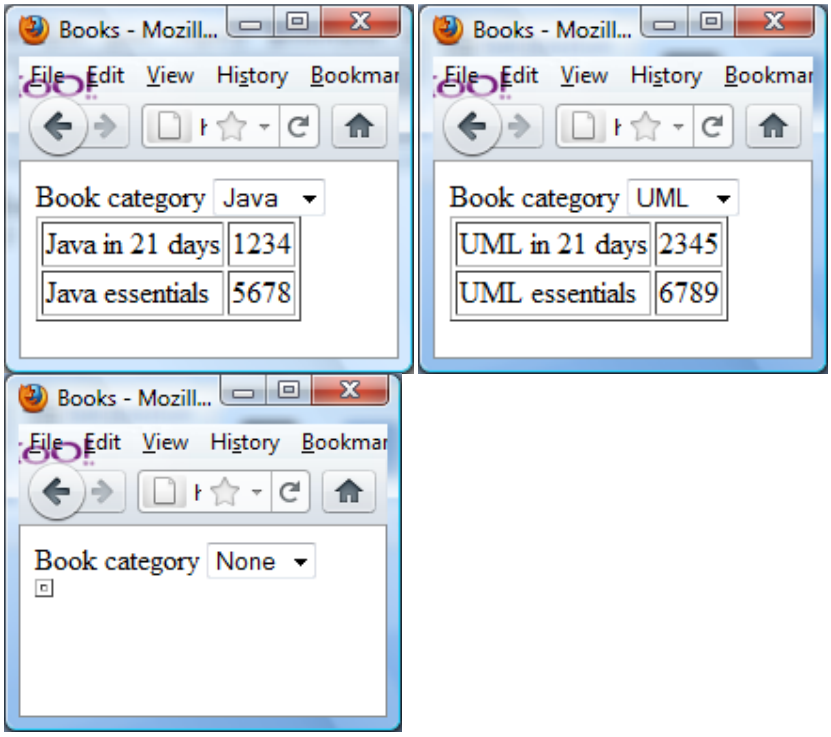
MyTime.java

```
public class MyTime {
  private int hours;
  private int minutes;

   //getters and setters are not shown
}
```

TimeConverter.java

```
@FacesConverter( value="timeConverter" )
public class TimeConverter implements Converter {

   @Override
   public Object getAsObject(FacesContext context, UIComponent component, String value) {
      MyTime time = new MyTime();
      time.setHours(Integer.parseInt(value.substring(0,value.indexOf(":"))));
      time.setMinutes(Integer.parseInt(value.substring(value.indexOf(":")+1,value.length())));
      return time;
   }

   @Override
   public String getAsString(FacesContext context, UIComponent component, Object value) {
      MyTime time = (MyTime)value;
      return time.getHours()+" hours and "+time.getMinutes()+" minutes";

   }
}
```

Question 3 Events [20 points] {25 minutes}



Write an JSF application with the following behavior:
The application shows a selectOneMenu control with the values Java, UML and None.
If you select Java, then you see a table showing 2 Java books.
If you select UML, then you see a table showing 2 UML books.
If you select None, then the table is empty.
The first column in the table is the name of the book, and the second column is the ISBN number (in this example just a string containing 4 characters)

Complete the partial given code such that the application works with the given behavior. You only have to complete the code for the xhtml file and the managed bean.
IMPORTANT: do not write getter and setter methods!

books.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core">
    <h:head>
      <title>Books</title>
    </h:head>
    <h:body>
      <h:form>
    Book category
    <h:selectOneMenu value="#{bookbean.selectedBook}"
                valueChangeListener="#{bookbean.changeBook}"
                onchange="submit()">
      <f:selectItems value="#{bookbean.books}"/>
    </h:selectOneMenu>
    <br />
    <h:dataTable value="#{bookbean.booklist}" var="book" border="1" >
      <h:column>
        <h:outputText value="#{book.name}"/>
      </h:column>
      <h:column>
        <h:outputText value="#{book.isbn}"/>
      </h:column>
      </h:dataTable>
      </h:form>
    </h:body>
```

```
import java.util.*;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
import javax.faces.event.ValueChangeEvent;
import javax.faces.model.SelectItem;

@ManagedBean
@RequestScoped
public class Bookbean {


   private String selectedBook;
   private Collection<SelectItem> books = new ArrayList();
   private Collection<Book> booklist= new ArrayList<Book>();

   public Bookbean(){
      books.add(new SelectItem("None","None"));
      books.add(new SelectItem("Java","Java"));
      books.add(new SelectItem("UML","UML"));
   }

   public void changeBook(ValueChangeEvent valueChangeEvent) {
      if (valueChangeEvent.getNewValue().toString().equals("Java")){
         booklist.add(new Book("Java in 21 days", "1234"));
         booklist.add(new Book("Java essentials", "5678"));
      } else if (valueChangeEvent.getNewValue().toString().equals("UML")){
         booklist.add(new Book("UML in 21 days", "2345"));
         booklist.add(new Book("UML essentials", "6789"));
      } else {
         booklist.clear();
      }
   }



}
```

Question 4 [15 points] {15 minutes}

a. **Explain how the tag <ui:composition> is used within JSF. What is the purpose of this tag.**

<ui:composition template=*"template.xhtml"*>
This tag is used for defining templates for JSF facelets

b. **Explain for what purpose you need the @ManagedProperty annotation in JSF.**

A ManagedProperty is a property that is managed by JSF. JSF will instantiate or inject the correct class from the correct scope.

c. **Suppose we have the following JSF tag in our xhtml page:**

<h:inputText id="card" value="#{payment.card}" />

We want to add validation to this inputText. We want to validate that the user types at least 12 characters in this field. If the user does not type 12 characters, then we need to show an error message just after this inputText. Show the necessary tags we need to add this behavior. (Only show the necessary tags and their attributes, do not write any Java code)

```
<h:inputText id="card" value="#{payment.card}" />
    <f:validateLength maximum="12" minimum="12"/>
</h:inputText>
<h:message style="color: red" for="card"/>
```

Question 5 SCI [5 points] {10 minutes}
JSF 2.0 offers functionality for Dependency Injection. Describe how we can relate Dependency Injection within JSF to principles of SCI. Your answer should be about half a page, but should not exceed one page (handwritten). The number of points you get for this questions depend on how well you explain the relationship between Dependency Injection within JSF and the principles of SCI.