11th homework assignment; JAVA, Academic year 2011/2012; FER

As usual, please see the last page. I mean it! You are back? OK. Here we have 4 problems for you to solve.

Introduction

For this homework you will upgrade a simple web application described in text document we used during last lectures. Once done, you will prepare two files and upload them to Ferko:

- a ZIP archive of your eclipse project,
- a WAR archive of your finished web application that can be deployed and tested.

Problem 1.

If you still did not complete the simple web application described in text file we used during last lecture, complete it (see “Sekcija 5” in that document).

Problem 2.

You will implement a functionality to add new entries (modeled by domain class `Unos`) and edit existing ones. Proceed as follows.

Part 1

Create a class `UnosForm` (place it in a package `hr.fer.zemris.java.tecaj_13.webforms`). We will use this class for representing instances of `Unos` class in web forms. This class must mirror the `Unos` class; however, all properties must be of a type `String` (e.g. in `Unos` class you have `long id`, in `UnosForm` you will have `String id`). We will refer to these fields as main fields. For each declared field in class `UnosForm` declare a private member with a single underscore in front and with original property type; e.g. `long _id`). For these properties do not declare getters nor setters. We will refer to these fields as shadow fields. Declare also a private field `Map<String, String> errors`. We will use this map for associating user errors with each form field. Implement public method `hasError()` that return `true` if `errors` map is not empty and `false` otherwise. Implement public method `clearErrors()` that clears the `errors` map.

We will assume that each field that is declared in this class will be sent by browser using same-named parameters.

Now declare and implement following three methods in `UnosForm` class:

```
1. public void fill(HttpServletRequest request);
2. public boolean checkErrors();
3. public void fromDomainObject(Unos unos);
4. public void toDomainObject(Unos unos);
```

The method `fill(...)` will scan parameters present in `HttpServletRequest` and fill main fields with provided values. For example, if the part of URL containing parameters is:

```
?id=17&title=Aloha&message=Nebitna
```

this method will set `id`, `title` and `message` fields of `UnosForm` to provided values. All other main fields of `UnosForm` must be cleared. This method, hence, transfers data from `HttpServletRequest` into `UnosForm`.

Please observe that will always succeed since URL parameter as well as `UnosForm`'s main fields are `Strings` so no error can occur.

The method `checkErrors()` will try to convert main fields into shadow fields. Here an errors are possible and they must be handled as described next.

- For each shadow field of type `String` simply copy a value from corresponding main field. If corresponding field in domain object has some constraints (such as: can not be null or empty, can not be longer than some predefined length etc), check if these constraints hold on shadow field. If not, add appropriate error in errors map using main field property name as key.

- For each shadow field of other type than String try to perform a conversion; if successful, and if corresponding field in domain object has some constraints (such as: can not be null or empty, can not be longer than some predefined length etc), check if these constraints hold on shadow field. If conversion fails or there are unsatisfied constraints, add appropriate error in errors map using main field property name as key.

The result of `checkErrors` is `true` if at least one error occurred. If no errors occurred, method must return `false.`

Let's see and example. Suppose parameters in URL are:

```
?id=17x&title=&message=Nebitna
```

and we have an instance of `UnosForm u`. A call of method `fill` will set:

```
u.id=17x
u.title=""
u.message="Nebitna"
```

Now a call to `checkErrors` will be unable to set `u._id` since `17x` can not be converted to long so an error mapping {`"id" => "Not a valid number"`} will be placed in `errors` map. Also, since `u._title` will be empty string which is not permitted for `title`, an error mapping {`"title" => "Must be provided!"`} will be placed in `errors` map. Note: you can chose better error texts.

Method `fromDomainObject` will fill shadow fields based on content found in given `Unos` instance and then it will fill main fields (`String`-typed shadow fields will be simply copied, non-`String`-typed shadow fields will be converted into `String` representation). For conversion of `Date` objects to and from `String` use utility class `SimpleDateFormat` with constructor ("`yyyy-MM-dd HH:mm:ss`"); you will find it provides methods `parse` and `format.`

Method `toDomainObject` will fill given `Unos` instance with values found in shadow fields.

<u>Part 2</u>

Create a servlet `UnosServlet` and map it to a pattern "`/servleti/unos/*`". This will allow you invoke the servlet by any URL of the form:

```
http://localhost:8080/aplikacija3/servleti/unos
http://localhost:8080/aplikacija3/servleti/unos/XXX
```

See section 3.5 "Request Path Elements" in Java Servlet Specification PDF (servlet-3_0-final-spec.pdf) for instruction how to obtain and interpret the context path, servlet path. You will have to find out what path was given when your servlet was invoked. We will define three “legal” ways of calling this servlet (i.e. the XXX in above example will have three legal values), as follows.

- new – when servlet is called like this, you will create a new instance of UnosForm class (uf), create a new instance of Unos class (u) and call uf.fromDomainObject(u). Then you will bind uf into request attribute “model.object” and forward execution to JSP in location “/WEB-INF/pages/unos.jsp”.

- edit – when servlet is called like this, an extra parameter id must be given; you will ask your DAO implementation to fetch an Unos u with provided id from persistent storage. You will create a new instance of UnosForm class (uf) and call uf.fromDomainObject(u). Then you will bind uf into request attribute “model.object” and forward execution to JSP in location “/WEB-INF/pages/unos.jsp”.

- save – when servlet is called like this, you will create a new instance of UnosForm class (uf) and call uf.fill(request) to fill this form with received parameters. Then you will call uf.checkErrors() to fill shadow fields. If any error is reported beside error on id, you will forward execution to JSP in location “/WEB-INF/pages/unos.jsp” and exit. If there is ony error with field id and if field id is null (or empty), this means that user is trying to insert a new record. Now, if id is null or empty, create a new instance of Unos u, call uf.toDomainObject(u), access your DAO implementation and call save(u) – at the moment, you do not have this method so you will need to implement it as well. In case there was a valid id present in uf, you will access your DAO, try to fetch an Unos u with given id, and if such object is found, call uf.toDomainObject(u), access your DAO implementation and call update(u) – at the moment, you do not have this method so you will need to implement it as well. Once the save or update is completed, you must send the users browser a redirection command to your servlet “ListajKratko”. For this, you will use:

  ```
  resp.sendRedirect(location)
  ```

  By sending redirect you will prevent that when user presses “Refresh/Reload” in his browser, that editing/inserting action is repeated. Instead, only “redirection” will be repeated with no consequences for our data.

Page /WEB-INF/pages/unos.jsp is responsible for rendering a HTML form for a single Unos. This page will access UnosForm object stored in request's attributes under key “model.object” and it will use its main fields for form generation. For each UnosForm's fields appropriate FORM control must be generated and prefilled; additionally, if there is a registered error for that field, it must be displayed to user.

Please note: FORM's action will be something like “/aplikacija3/servleti/unos/save”. However, you are not allowed to hard-code this into JSP, so that application can be deployed under some other context without the need to modify all JSP-s. Generate this URL on the fly (perhaps even in servlet, and then register it as some temporary attribute that will JSP access).

Important: create a new class HTMLSupport in package hr.fer.zemris.java.tecaj_13.web. Implement two static methods: escapeForTagAttribute and escapeForHTMLBody.

```
String escapeForTagAttribute(String text);
```

This method must return empty string if text is null. Otherwise, it must create an escaped version of given text in which all '"' are replaced by “\"”, all '' are replaced by “\'”, all ascii 10 are replaced by “\n” and all ascii 13 are replaced by “\r”.

```
String escapeForHTMLBody(String text);
```

This method must return `empty` string if text is `null.` Otherwise, it must create an escaped version of given text in which all '<' are replaced by “`&lt;`”, all '>' are replaced by “`&gt;`”, all “&” are replaced by “`&amp;`”.

Use this functions in JSP to escape initial values of fields. The title of JSP should indicate if you are editing or creating a new record.

Problem 3.

Modify the HTML resulting from servlet `ListajKratko` to include a link for each record to a servlet `UnosDetails` mapped by `/servleti/details`. Clicking on this link a new page must be displayed with full details on selected record (i.e. you must display all fields for that record). On this page include a link that will allow user to edit this record (you developed the required functionality as part of Problem 2). Now modify the redirecting destination of save action to `/servleti/details` for this record.

Modify the HTML resulting from servlet `ListajKratko` to include a single link that will allow user to add new record. You developed the required functionality as part of Problem 2.

Problem 4.

Modify the HTML resulting from servlet `ListajKratko` to include a single link that will dynamically generate a PDF document containing detailed information for all records. You will fetch all records from DAO and generate a PDF with one page for each record. On each page you will display all data for corresponding record.

For PDF creation you will use *iText* library which is free, open source, stable, proven and popular Java library for PDF creation and manipulation (it is even ported to .NET platform) available on:

http://itextpdf.com/

Please note. You can consult with your peers and exchange ideas about this homework *before* you start actual coding. Once you open you IDE and start coding, consultations with others (except with me) will be regarded as cheating. You can not use any of preexisting code or libraries for this homework (whether it is yours old code or someones else). Document your code!

In order to solve this homework, create a blank Eclipse Java Project and write your code inside. Once you are done, export project as a ZIP archive and upload this archive on Ferko before the deadline. Do not forget to lock your upload or upload will not be accepted.