logMeIn

Table of contents

1. Jags Script

```
# eg/logMeIn.jags

## set of user logins, with password and email address for each

User login:userName-> password email:emailAddr
(userName, password=)
[*] logout
```

2. Application Description

To the user, JAppGen is a set of two screens. The first asks the user to input a user name and password and has a **submit** button. If the user inputs an existing user name, one which is in the database, and the correct password, the user passes to the second screen, which displays what the database knows about the user (user name, password, and email address) and a button labelled **logout**. When the user presses the button, the first screen appears again.

JAppGen generates the entire application from the script above. Lines beginning with a sharp sign (#) are comments. The double sharp signs (##) are intended to function as Javadocs do.

The first line of the script,

```
User login:userName-> password email:emailAddr
```

declares a class `User` which has three fields: `userName, password,` and `emailAddr.` Two of these have labels which are different from the field name. The label precedes the field name and is separated from it by a colon (':'). The label on `userName` is `logIn`. The label on `emailAddr` is `email`. Labels are used in screen generation.

Each field has a type; the default type corresponds to a Java String. Each of `User`'s fields is a JAppGen built-in type. `userName` is a string consisting of six to twenty alphanumeric characters, where we consider letters, digits, and the underscore character ('_') to be alphanumeric. Note that no delimiters (space, tab, newlines, etc) are permitted in `userName`s.

For our purposes, `password` is much the same as a `userName`, with one difference: it is not echoed on input. So passwords consist of six to ten alphanumeric characters and may not contain delimiters.

An `emailAddr` is an RFC 822 email address. We require that this be at least 6 characters in length and no more than 255. `a@b.dk` is an example of the shortest possible `emailAddr`. Early implementations of JAppGen will not accept quoted names or angle brackets around the email address.

The arrow ('->') following `userName` signifies that it is a key field, in this case the primary key.

The second line of the script,

```
(userName password=)
```

declares a screen. Because the definition is parenthesized, contained between '(' and ')', it is a form. Because the form accepts not just the key, `userName`, but also a second field name followed by an equal sign ('='), it is specifically a validation form. This means that the application will accept values for both fields, validate them, then retrieve a record from the database using the primary key. If the object retrieved from the database matches the values entered (`userName` and `password`) the form succeeds and, in this case, passes to the next screen. Otherwise it fails. This associates an error message with one or more fields and reinvokes the same screen.

By default a form is a three column screen. The first column consists of right-justified labels. The second column contains left-justified values or input fields. In this case both cells are input fields. The third column contains any error message associated with the form fields. Initially there are no error messages. Thereafter if the form is reached because of an error, the messages are displayed, but if the form is reached after a success, the error message hash is cleared.

The third line

```
[*] logout
```

declares the second screen. Because the definition is bracketed, contained between '[' and ']', it is a table; by default this is a two-column table. The first column contains right-justified labels and the second column left-justified associated values. A list of fields to be displayed is contained within the brackets. In this case the list is just a star ('*'), meaning all of the fields. Because no specific class is mentioned, these are the fields in the default class, the last class declared.

The table definition is followed by a button label, **logout**. If there were no such label, the button would be labeled **submit**.

Terseness is valued in JAppGen. A more verbose version of the same script is

```
     test::
     User login:userName:userName->User  \
          password:password:password     \
          email:emailAddr:emailAddr
     Login  (User.userName, User.password=) submit => Logout
     Logout [User.userName, User.password, User.emailAddr] logout => Login
```

The database name, `test`, precedes the double colon ('::') in the first line. This is the default database name. The next three lines declare the `User` class, in each case specifying first the label on the field, then the field name, then its type. The `->User` explicitly declares that `userName` is the primary key to the database table corresponding to the class `User`.

The next line assigns a name to the validation form, `Login` and then explicitly state that each field is a field of the `User` class. The default label on the button, **submit**, is declared, and the transition on success (the '=>') is explicitly declared to be to the screen `Logout`; this is the default, as it is the next screen in sequence.

The last line similarly explicitly labels the table, names the class fields are members of, and declares that the transition on success is to the first screen, as is the default.

3. Generated Code

3.1. Front End

The JAppGen front end is based on JavaServer Faces and so follows the model-view-controller paradigm.

3.2. Presentation/Object Mapping

3.3. Application Object Model

3.4. Object/Relational Mapping

3.5. SQL Database