Exercises: Managed Beans I

Make a new Dynamic Web Project based on the jsf-blank project or on your project from the previous exercises. If you copy an existing project, don't forget to update the new project as described earlier (go to the file system or Eclipse Navigator and edit *eclipseWorkspace*/*yourNewProject*/.settings/org.eclipse.wst.common.component and change the instances of the old project name to the new one). Close the Navigator when done.

Again, remember that there is a "samples" folder inside of jsf-blank that has templates of the very basic layout of JSF pages. You can copy and rename one of those template files and use it as a starting point for your .xhtml files.

The most important problem is #2: using business logic, so that your output page shows something that was not supplied by the user, but rather was calculated based on the user input. The first problem is a warmup along the way to that goal.

1. Make a form to gather an employee name, employee ID, and the name of a health plan to sign up for. If all three values are present (i.e., are something other than empty strings), display a confirmation page that says "You are registered for the health plan" and that shows the name, ID, and health plan. If any of the inputs are missing, display a page that says "Missing Input Data".

   Note that you should make the employee ID be a String, not an int. JSF can do type conversion, but if you make use of this capability, you need to handle the case when the user enters something in an illegal format, and we won't know how to do that until the validation lectures. So, for now, just make the employee ID a String (e.g. "a1234" or simply "1234" as a String, not an int).

2. Make a new form that works similarly to the old one. This time, however, verify that the health plan entered is really one of the available health plans. Give an error message (e.g., "CMS Prime is not an available health plan") for unknown health plans. In the confirmation page, also show the monthly premium and contact phone number for that health plan (in addition to the data shown earlier). This requirement means that you need some simple business logic to map a health plan name to a HealthPlan object. To simplify this part of the exercise, feel free to steal HealthPlan (represents a plan), HealthPlanFinder (interface for finding a plan from a name), and SimpleChoicesHealthPlanFinder (concrete version of interface that has a few hard-coded plans) from my managed-beans-1-exercises project.