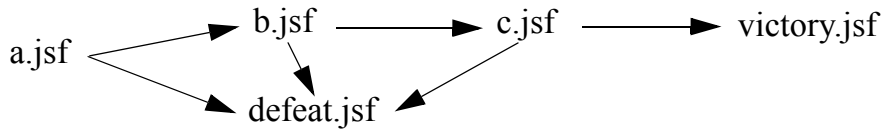
Exercises: JSF 2 Programming Basics

For these exercises and later ones, note that there is a "samples" folder inside of jsf-blank that has templates of the very basic layout of JSF pages. So, copy and rename one of those template files and use it as a starting point for your .xhtml files.

1. Make a new Dynamic Web Project based on jsf-blank. Remember that there are two different ways of making a JSF 2.0 project: copying/pasting jsf-blank, or making a blank project and copying the pieces from jsf-blank. If you do the copy/paste approach, you also have to go to the file system (or Eclipse Navigator) and edit *eclipseWorkspace*/*yourNewProject*/.settings/org.eclipse.wst.common.component. Close the Navigator when you are done. This copy/paste approach is perhaps more trouble than it is worth for this exercise, but it is better in the long run if you have Eclipse 3.6+ because you copy all the project code with it, and because you get the JSF 2 project facet, so Eclipse will give help and show previews when editing .xhtml files (now) and help you in editing faces-config.xml (in later exercises). Also, remember that if you use JBoss 6+, Glassfish 3+, or another Java EE 6 server, you should delete the JAR files from WEB-INF/lib because the classes defined in those JARs are already built into the server. Finally, since several of these exercises are similar to examples in the "basics" project, feel free to copy and adapt code from that project.

2. Implement the following navigation flow. You start at a.jsf, press a button, and randomly get either b.jsf or defeat.jsf. From page b, after pressing a button you randomly get c.jsf or defeat.jsf. From page c, after pressing a button you randomly get victory.jsf or defeat.jsf. Note that since JSF navigation is based on forwarding, not redirects, when you press a button, the URL of the *previous* page is shown in the browser.



Try building up incrementally, so start by making a.jsf that shows a button. Then, make it so that clicking the button on page a gives you either b.jsf or defeat.jsf. Then, add a button to page b that has similar behavior. Etc. Also, note that with what we know so far, there is no way to avoid writing three separate methods (or classes): one for each of the push buttons. Finally, there is no need to use my fancy RandomUtils for this example: to flip a coin as in this case, just call Math.random() and compare the output to 0.5.

3. Make a page that collects a first name and a last name. If either the first or last name is missing (i.e., an empty string), show an error page. If both the names are there, go to a results page that shows the first and last names in a bulleted (<ul>) list.