Introduction to Homework 3

There are two parts to homework 3. One part of the homework is the coding project described here. The other part is the product proposal described in an accompanying write up. You will be working in pairs on both parts of this homework.

The purpose of this coding project is to help you get started using the network to communicate between midlets and servlets.

Overview of the coding project

There are two zip files for this project, one for the midlet side and one for the servlet side. The basic idea is that you will write a midlet that reads and displays information provided to it by a servlet. (You may recognize this as a small-scale version of the architecture that I am proposing for the major class projects.)

The midlet zip contains the m3 directory. This directory contains the files necessary to build a midlet suite and package it in the m3 jar and jad files needed to run in the emulator. As provided to you, there is one midlet in the suite: UView. It reads a list of available URLs defined in the jad file, and then provides a list of items on the display that connect to those URLs. Selecting an item retrieves information from the associated URL. In this case, the information retrieved is somewhat low value; it's the same set of razor manufacturer names as in hw2. Initially, the midlet reads from a servlet that is installed under my name on the server, so you can use your midlet without having to do anything on the server side.

Your first task is to write a midlet that retrieves the list of available URLs from the web instead of from the jad file.

The servlet zip contains the s3 directory. This directory contains the files necessary to build a web application and package it in the war file for installation and running in Tomcat. As provided to you, there are two servlets in the web application: Razor and Contexter. Razor is an implementation of the simple text output servlet from hw2. Contexter reads and displays various elements of its environment on a web page, including an initial parameter specified in the web application deployment descriptor (wadd) file.

Your second task is to write a servlet that generates the list of available URLs and makes it available to your midlet written earlier.

When these two functions are working, you will have moved the responsibility for knowing the list of available URLs out of the midlet and put it on the servlet side of the architecture. Deciding where to locate this knowledge and how to update it will be an architectural issue for the class projects.

Preparation

1. Verify that the various parts and pieces of the course software are available on your development system, either in the lab or elsewhere. If there are any problems let us know right away. Of particular importance is the new version of the ant tasks for building the midlet suite: antenna-bin-r3.jar. This must be installed, replacing the old version. Instructions for this are on the class software page.

2. Download the zip files for this homework from the web site. Unzip them to your development directory. You will have two top-level directories at this point: m3 and s3. The contents of these directories will be familiar to you from the two previous homeworks although there are slight differences in the build files and the directory names in some places.

3. The build.xml files in particular have been modified to be closer in layout and content. In both cases there is a "build" target to compile and package the midlet suite (jar/jad) or the web application (war) and put the binary files in the dist directory. Also, both files include a "dist" target that does the build and also creates and copies the documentation to the dist directory. The server side Ant file has been renamed manage.xml and lives in the same top-level directory as build.xml.

4. Each of the three Ant files has an associated properties file. Remember to set wtk.home and catalina.home correctly for your development system.

5. The midlet jad file is in m3/src. This file is copied to the dist directory every time you build the midlet suite. Don't edit the copy in dist; it gets overwritten every build.

Assignment – Midlet NetGet in m3

There is one midlet provided to you (UView); your task is to write a second one (NetGet) that implements the capability of getting the list of URLs from the net instead of from the application properties defined in the jad file. You should add the new midlet to the m3 midlet suite.

1. The UView midlet reads a list of labels and URLs from the application properties, then creates a MenuPage containing a link for each item. Each item on the menu page points to a URLPage. When the URLPage loads, it starts a second thread that actually does the reading. It is important that time consuming activities like a web connection be done in a thread separate from the user command thread to maintain responsiveness.

2. The MenuPage class is part of the simple menu package, somewhat upgraded since you saw it first in hw 1. The sources for this package are provided to you in the src/menu directory. You are not required to use this package in your midlet; however, you will find it easier than starting from scratch. The "store" package is also included for completeness, but you don't need it for this assignment.

3. Read the UView source and the m3.jad file for an example of how to define and retrieve application properties in a midlet.

4. Read the URLPage source for an example of how to implement a second thread. In particular, notice that this class extends Page and implements the Runnable interface. Look at the code in Page.java for more parts of the multi-threading implementation. Note that the methods and code blocks that might cause changes that would affect the other thread are "synchronized" so that one thread cannot change something unexpectedly while another thread is running.

5. Now implement the NetGet application. The basic structure of the NetGet application is the same as UView. It creates a top-level menu page with child pages that are URLPages. However, the top-level menu page cannot be a plain MenuPage, because we need to read the list of URLs from the web. This requires a MenuPage subclass that has a separate input thread. I called this class ActiveMenuPage in my implementation. You need to implement both NetGet and ActiveMenuPage.

a. The main class of the new application is NetGet, as specified in the jad file. You can change this if you like, but that is what it is when delivered to you. You can copy UView.java to NetGet.java and just chop it up to get started on NetGet.

b. NetGet should read the value of the application property Blade-List-URL to find out the URL where the list of URLs is available. This is initially defined in the jad file as:

```
http://cubist.cs.washington.edu:8080/djohnson/list
```

This is the "source URL" that should be passed to the ActiveMenuPage so that it can build the list of URLPages.

c. If you look at the output of the URL given above, you will see that it is something like this:

```
Device 1$http://cubist.cs.washington.edu:8080/djohnson/device?index=1
Device 2$http://cubist.cs.washington.edu:8080/djohnson/device?index=2
```

This is the simple list of item names and associated URLs that ActiveMenuPage reads to build its display list and linked pages.

d. Your implementation of ActiveMenuPage will probably be easiest if you extend the MenuPage class. Take a close look at MenuPage, since it provides most of the methods and instance variables needed for ActiveMenuPage. The run() method and its support classes and instance variables that you need to add can be based on those in URLPage.

e. The run() thread in ActiveMenuPage reads the list of URLs from the source URL it is given. It parses those lines and builds a new URLPage for each line. The URLPages are added as children to the menu page, then the run() thread exits. In my implementation the page is completely rebuilt every time the menu page is reloaded. Although this would be costly on an actual device, it is acceptable for our purposes right now.

f. The run() method is in charge of setting the commands that are visible during its operation. It's nice if the user has the option to cancel long running operations. Also, the thread should provide status information to the user as the input operation is proceeding. Remember that any changes in the display should be synchronized and the thread must check for a request for exit before it makes changes. Again, see the run() method and its support methods in URLPage for an example.

g. When you get done with this part, write a brief release-notes.txt file and store that in the m3/doc directory. In the release notes, include a brief description of the features of your midlet. Also describe any known bugs or limitations.

Assignment – Servlet Lister in s3

There are two servlets provided to you in the web application: Razor and Contexter; your task is to write a third one (Lister) that provides the list of URLs to the NetGet midlet. You should add the new servlet to the s3 web application.

1. The Lister servlet reads a list of labels and URLs from the application init parameters. The parameters are defined in the web.xml file, as shown in the entry for Contexter.

2. When an HTTP GET request arrives, Lister formats the information as shown in the midlet discussion above. The format is <item display name> $ <associated URL> <new-line>. The mime content type of the response is text/plain.

3. The URLs that it provides to NetGet should point to your web application, not mine. So when you update web.xml to define the parameters, you will have entries like this:

```
<init-param>
    <param-name>Blade-Name-1</param-name>
    <param-value>Device 1</param-value>
</init-param>
<init-param>
    <param-name>Blade-URL-1</param-name>
    <param-value>
        http://cubist.cs.washington.edu:8080/youraccount/device?index=1
    </param-value>
</init-param>
```

I called the parameters Blade this-and-that, but you can name them anything you want.

4. The path "/device" is mapped to your Razor servlet and the path "/list" is mapped to your Lister servlet in servlet mapping entries in web.xml.

5. Build and install your web application on Tomcat. Verify that it is producing the expecting outputs by connecting to the various URLs directly with a browser.

6. Once you decided that the servlet is working correctly, go back and change the source URL in the jad file for NetGet to point to your /list servlet path instead of mine. If all is well, you should be able to connect from the midlet and see the list show up there that you have defined on the server, and then connect to the various little information sources.

7. When you get done with this part, write a brief release-notes.txt file and store that in the s3/doc directory. In the release notes, include a brief description of the features of your servlet. Also describe any known bugs or limitations.

8. I know that razor manufacturer names are not the most exciting factoids to be retrieving! Defining other more interesting possibilities is the next phase of the homework. You get to dream up the various info sources and figure out how we might use them. Onward and upward!

Turn in

Build two zip files that contain all the files in your directories and have your user name in the file name. Using Ant, the commands

```
s3> ant -Dusername=youraccount archive
s3> cd ../m3
m3> ant -Dusername=youraccount archive
```

will build the archive files in a zips directory at the same level as m3 and s3. Verify that the archives do indeed have all your files in them, then follow the turnin link on the web site.

Remember that there is another part to this homework: the project concepts. There is a file to be turned in for that too.

This homework is due before midnight, Tuesday January 28. Don't be late, the turnin server closes automatically!