Using the New java.io.Console Class

Published by Tony at 10:08 pm under Java

The 1.6 release of the JDK included a new java.io.Console class, which adds some new features to enhance and simplify command-line applications. Notably, `Console` includes a method specifically for reading passwords that disables console echo and returns a `char` array; both important for security.

Scenario: Getting Username and Password
Getting a user's username and password is probably one of the more common uses of the `Console` class. It's fairly simple to do, but there are some things to look out for. Before you do anything else, you have to get a `Console` instance. Looking at the API, you'll notice that `Console` has no public constructors. In fact, the only way to get a `Console` instance is by calling the `System` method:

```
Console con = System.console();
```

Simple enough, right? There is something you need to watch out for, however: The `System.console()` method can return `null`. According to the API:

> If the virtual machine is started from an interactive command line without redirecting the standard input and output streams then its console will exist and will typically be connected to the keyboard and display from which the virtual machine was launched. If the virtual machine is started automatically, for example by a background job scheduler, then it will typically not have a console.

So, all you have to do is start it interactively, right? Well, sort of. This is the way it **should** work, but currently Eclipse (and some other IDEs) don't yet support this feature "properly", so if you try to run a program that calls `System.console()` from *within* your favorite IDE, it may return `null`. You have been warned.

The following code demonstrates one method for getting login information.

```
...
private static final int MAX_LOGINS = 3;
...
public boolean login()
{
  Console con = System.console();
  boolean auth = false;

  if (con != null)
  {
    int count = 0;
    do
    {
      String uname = con.readLine("Enter your username: ");
      char[] pwd = con.readPassword("Enter %s's password: ", uname);
      auth = authenticate(uname, pwd); // authenticate login info
      Arrays.fill(pwd, ' '); // delete password from memory
      con.writer().write("\n\n");  // output a couple of newlines
    } while (!auth && ++count < MAX_LOGINS);
  }
  return auth;
}
```

The first thing I'm doing here is getting a `Console` instance and checking to make sure it's not `null` before proceeding. Once I have a `Console` instance, I can use the `readLine` and `readPassword` methods to get the login information. Both of these methods are overloaded, and include the following overloads:

`public String readLine()`
Reads a single line of text from the console.

`public String readLine(String fmt, Object... args)`
Provides a formatted prompt, then reads a single line of text from the console.

`public char[] readPassword()`
Reads a password or passphrase from the console with echoing disabled

`public char[] readPassword(String fmt, Object... args)`
Provides a formatted prompt, then reads a password or passphrase from the console with echoing disabled.

The versions I'm using combine prompting the user and getting input into one. If you really wanted to separate the two, you could use the `Console.printf` method to output your prompt and one of the no-arg methods above to get user input. That's not necessarily *incorrect*, but definitely more verbose.

The `readLine` method takes a formatted String and a variable list of arguments; exactly like the `System.out.printf` method. The `readLine` method returns a line of input from the console, as a `String`, not including line termination characters. It returns `null`, if the end of the stream has been reached. The `readPassword` method is almost identical in functionality, except that it returns a `char` array of the user input. You'll also notice that it disables console echo, so someone walking by can't see what's being typed.

Once I have the login information, I'm using a method called `authenticate` to check it. The implementation of this method is outside of the scope of this post. You could stub it out and test it, using hard-coded values.

Notice that, as soon as I'm done using the password information, I blank it out using the `Arrays.fill` utility method. This will minimize the lifetime of sensitive (password) data in memory.

Incidentally, I've also wrapped all of this in a loop that will run until the `authenticate` method returns `true` or until `count` is equal to `MAX_LOGINS`.

The Console class also has a few other methods, but the above four are the most interesting and likely to be the most-used. You might also use either the `printf` or `format` methods. According to the spec, these methods both behave identically to each other and exactly like `System.out.printf`. If you want to get the `PrintWriter` associated with this console, you can use the `writer()` method. Similarly, the `reader()` method retrieves the `Reader` object. Last but not least is the `flush()` method, which is a must since `Console` implements `Flushable`. Nothing surprising or groundbreaking with these last few methods, but I'd be remiss if I didn't at least mention them.