```java
// joi/10/juno/LoginInterpreter.java
//
//
// Copyright 2003 Ethan Bolker and Bill Campbell

import java.util.*;

/**
 * Interpreter for Juno login commands.
 *
 * There are so few commands that if-then-else logic is OK.
 *
 * @version 10
 */

public class LoginInterpreter
    implements InterpreterInterface
{
    private static final String LOGIN_COMMANDS =
        "help, register, <username>, exit";

    private Juno system;             // the Juno object
    private OutputInterface console; // where output goes

    /**
     * Construct a new LoginInterpreter for interpreting
     * login commands.
     *
     * @param system the system creating this interpreter.
     * @param console the Terminal used for input and output.
     */

    public LoginInterpreter( Juno system, OutputInterface console)
    {
        this.system  = system;
        this.console = console;
    }

    /**
     * Set the console for this interpreter.  Used by the
     * creator of this interpreter.
     *
     * @param console the Terminal to be used for input and output.
     */

    public void setConsole( OutputInterface console)
    {
        this.console = console;
    }

    /**
     * Simulates behavior at login: prompt.
     */

    public void CLILogin()
    {
```

```java
        welcome();
        boolean moreWork = true;
        while( moreWork ) {
            moreWork = interpret(((InputInterface)console).
                                     readLine( "Juno login: " ) );
        }
    }


    /**
     * Parse user’s command line and dispatch appropriate
     * semantic action.
     *
     * @param  inputLine the User’s instructions.
     * @return true except for "exit" command
     *         or null inputLine.
     */

    public boolean interpret( String inputLine )
    {
        if (inputLine == null) {
            return false;
        }
        StringTokenizer st =
            new StringTokenizer( inputLine );
        if (st.countTokens() == 0) {
            return true; // skip blank line
        }
        String visitor = st.nextToken();
        if (visitor.equals( "exit" )) {
            return false;
        }
        if (visitor.equals( "register" )) {
            register( st );
        }
        else if (visitor.equals( "help" )) {
            help();
        }
        else {
            String password;
            try {
                if (console.isGUI()) {
                    password = st.nextToken();
                }
                else {;
                    password = readPassword( "password: " );
                }
                User user = system.lookupUser(visitor);
                user.matchPassword( password );
                new Shell( system, user, console );
            }
            catch (Exception e) {

                // NullPointerException if no such user,
                // JunoException if password fails to match -
                // message to user doesn’t give away which.
                // The sysadmin would probably want a log
                // that did keep track.
                //
                // Other exceptions should be caught in shell()

                console.println("sorry");
            }
        }
        return true;
    }

    // Register a new user, giving him or her a login name and a
    // home directory on the system.
    //
    // StringTokenizer argument contains the new user’s login name
    // followed by full real name.

    private void register( StringTokenizer line )
    {
        String username = "";
        String password = "";
        String realname = "";
        try {
            username = line.nextToken();
            password = line.nextToken();
            realname = line.nextToken("").trim();
        }
        catch (NoSuchElementException e) {
        }

        if (username.equals("") || password.equals("")
            || realname.equals("") ) {
            console.println(
                  "please supply username, password, real name");
            return;
        }
        User user = system.lookupUser(username);

        if (user != null) {  // user already exists
            console.println("sorry");
            return;
        }
        if (badPassword( password )) {
            console.println("password too easy to guess");
            return;
        }
        Directory home = new Directory( username, null,
                                        system.getUserHomes() );

        user = system.createUser( username, home, password, realname );
        home.setOwner( user );
    }

    // test to see if password is unacceptable:
    //   fewer than 6 characters
    //   contains only alphabetic characters
```

```

    private boolean badPassword( String pwd )
    {
        if (pwd.length() < 6) {
            return true;
        }
        int nonAlphaCount = 0;
        for (int i=0; i < pwd.length(); i++) {
            if (!Character.isLetter(pwd.charAt(i))) {
                nonAlphaCount++;
            }
        }
        return (nonAlphaCount == 0);
    }

    // Used for reading the user’s password in CLI.

    private String readPassword( String prompt )
    {
        String line =
            ((InputInterface) console).readLine( prompt );
        StringTokenizer st = new StringTokenizer( line );
        try {
            return st.nextToken();
        }
        catch ( NoSuchElementException e )  {}
        return ""; // keeps compiler happy
    }


    // Display a short welcoming message, and remind users of
    // available commands.

    private void welcome()
    {
        console.println( "Welcome to " + system.getHostName() +
                         " running "   + system.getOS() +
                         " version "   + system.getVersion() );
        help();
    }

    // Remind user of available commands.

    private void help()
    {
        console.println( LOGIN_COMMANDS );
        console.println("");
    }
}
```