CS c352: Homework 1

Leo Przybylski
przybyls@arizona.edu

September 1, 2011

The write command

1. Task: Were starting to learn some shell commands and UNIX utilities. One we wont be talking about in lecture is write. Your task: Using your recently-acquired knowledge of how to learn about new commands, learn enough about write to answer each of the following questions:

a) What does write allow a user to do?

Communicate with other users on the same system via copying lines from one terminal to another.

b) When a user is done using write, how can the program be terminated? Specically, we want to know the character, not the name of the functionality associated with the character.

The character is called the end-of-file character or EOF. This can be applied using Ctrl-d keystroke.

c) How can write be used to connect with a user using a specic TTY?

The name of the tty can be specified on the command line as an optional argument to the write command following the user argument. For example:

write user [ttyname]

d) How can a user nd the TTY(s) of those people logged into the computer?

User TTYs can be discovered using the **who** and **w** commands.

e) What can a user to do protect herself from write? We want to know the exact command, including any necessary option(s)?

Just restrict access to the terminal for the tty group.

Listing 1: trnapp-config.xml

```
chmod 600 /dev/<ttyname>
```

Sorted Unique Argument List

Listing 2: trnapp-config.xml

```
/*
 * Copyright 2011 Leo Przybylski
 *
 * Licensed under the Educational Community License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.opensource.org/licenses/ecl2.php
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/*==============================================================================
 |    Assignment:  Prog1c
 |
 |        Author:  Leo Przybylski
 |      Language:  przybyls@lectura:~$ java -version
 |                 java version "1.6.0_26"
 |                 Java(TM) SE Runtime Environment (build 1.6.0_26-b03)
 |                 Oracle JRockit(R) (build R28.1.4-7-144370-1.6.0_26-20110617-2130-linux-
    x86_64, compiled mode)
 |    To Compile:  cd java;
 |                 javac edu/arizona/cs352/Prog1c.java
 |
 |         Class:  Csc352 Systems Programming and UNIX
 |    Instructor:  Dr. Lester McCann
 |      Due Date:  September 01, 2011 at 5pm
 |
 +------------------------------------------------------------------------------
 |
 |   Description:  Learn to handle command line arguments from a Unix shell. The
 |                 program parses arguments and sorts them ignoring leading hyphens
 |
 |         Input:  The program takes any number of arguments that may or may not be
 |                 prefixed with one or more hyphens.
 |
 |        Output:  An ascending sorted list of arguments passed in is printed out.
 |
 |     Algorithm:  The approach used is to iterate over args passed into the main
 |                 method and create a collection of arguments. A Comparator
 |                 is applied to ignore leading '-' and '--'.
 |
 |    Required Features Not Included:
 |
 |    Known Bugs:
 |
 *==============================================================================*/
package edu.arizona.cs352;

import java.util.Comparator;
import java.util.List;

import static java.util.Arrays.asList;
import static java.util.Collections.sort;

/**
 *
```

```
61  * <h3>Description</h3>
62  * Learn to handle command line arguments from a Unix shell. The
63  * program parses arguments and sorts them ignoring leading hyphens
64  *
65  * <h3>Input</h3>
66  * The program takes any number of arguments that may or may not be
67  * prefixed with one or more hyphens.
68  *
69  * <h3>Output</h3>
70  * An ascending sorted list of arguments passed in is printed out.
71  *
72  * <h3>Algorithm</h3>
73  * The approach used is to iterate over args passed into the main
74  * method and create a collection of arguments. A Comparator
75  *                is applied to ignore leading '-' and '--'.
76  *
77  * @author Leo Przybylski (przybyls [at] arizona.edu)
78  */
79 public class Prog1c {
80     private List<String> args;
81
82     /*-------------------------------------------------------- main -----
83       |  Function main
84       |
85       |  Purpose:  Entry point for the application. Handles parsing of arguments
86       |            and creates the Prog1c instance. Then prints the {@link String}
87       |            value of the {@link Prog1c} instance. In here a {@link Comparator}
88       |            is applied to the arguments to produce a sorted {@link List}
89       |
90       |  Parameters:
91       |      args (IN) -- arguments passed in from the commandline
92       |
93       |  Returns:  returns nothing.
94       *-----------------------------------------------------------------*/
95     public Prog1c(final String ... args) {
96         this.args = (List<String>) asList(args);
97         sort(getArgs(), new Comparator<String>() {
98
99                 /*-------------------------------------------- compareTo -----
100                  |  Function compareTo
101                  |
102                  |  Purpose:  Used when sorting the arguments {@link List} instance.
103                  | Compares two {@link String} instances. The {@link String} instances may
                        or
104                  | may not have preceeding '--' or '-'. When they do, these are stripped
                        from the
105                  | {@link String} instance. A copy is made first, so the original is not
                        effected.
106                  |
107                  |  Parameters:
108                  |      a - a String to compare
109                  |      b - another String to compare
110                  |
111                  |  Returns:  0 if same, -1 if a is smaller than b, and 1 if a is larger
                        than b
112                  *-----------------------------------------------------------------*/
113                 /**
114                  * Used when sorting the arguments {@link List} instance.
115                  * Compares two {@link String} instances. The {@link String} instances may or
116                  * may not have preceeding '--' or '-'. When they do, these are stripped from
                        the
117                  * {@link String} instance. A copy is made first, so the original is not
                        effected.
118                  *
119                  * @return 0 if same, -1 if a is smaller than b, and 1 if a is larger than b
120                  */
```

```
121                 @Override
122                 public int compare(final String a, final String b) {
123                     int preidx = 0;
124                     if (a.startsWith("--")) {
125                         preidx = "--".length();
126                     }
127                     else if (a.startsWith("-")) {
128                         preidx = "-".length();
129                     }
130
131                     final String stripped1 = a.substring(preidx);
132
133                     preidx = 0; // reset
134                     if (b.startsWith("--")) {
135                         preidx = "--".length();
136                     }
137                     else if (b.startsWith("-")) {
138                         preidx = "-".length();
139                     }
140
141                     final String stripped2 = b.substring(preidx);
142
143
144                     return stripped1.compareTo(stripped2);
145                 }
146             });
147     }
148
149     /**
150      * Gets the args attribute.
151      * @return Returns the args.
152      */
153     public List<String> getArgs() {
154         return args;
155     }
156
157     /**
158      * Sets the args attribute value.
159      * @param args The args to set.
160      */
161     public void setArgs(List<String> args) {
162         this.args = args;
163     }
164
165     /*------------------------------------------------------- main -----
166       |  Function main
167       |
168       |  Purpose:  Entry point for the application. Handles parsing of arguments
169       |            and creates the Prog1c instance. Then prints the {@link String}
170       |            value of the {@link Prog1c} instance.
171       |
172       |  Parameters:
173       |      args (IN) -- arguments passed in from the commandline
174       |
175       |  Returns:  returns nothing.
176       *-----------------------------------------------------------------*/
177     /**
178      * Iterates over args and creates a space-delimited {@link String} instance
179      *
180      * @return space-delimited {@link String} instance of a sorted {@link List} of {@link
           String} instances
181      */
182     @Override
183     public String toString() {
184         if (args == null || args.size() < 1) {
185             return "";
```

```
186         }
187
188         final StringBuilder retval = new StringBuilder();
189
190         for (final String arg : getArgs()) {
191             retval.append(arg).append(" ");
192         }
193
194         return retval.toString();
195     }
196
197     /*------------------------------------------------------- main -----
198       |  Function main
199       |
200       |  Purpose:  Entry point for the application. Handles parsing of arguments
201       |            and creates the Prog1c instance. Then prints the {@link String}
202       |            value of the {@link Prog1c} instance.
203       |
204       |  Parameters:
205       |      args (IN) -- arguments passed in from the commandline
206       |
207       |  Returns:  returns nothing.
208       *-----------------------------------------------------------------------*/
209     /**
210      *  Entry point for the application. Handles parsing of arguments
211      *            and creates the Prog1c instance. Then prints the {@link String}
212      *            value of the {@link Prog1c} instance.
213      *
214      * @param args is an array of {@link String} instances representing arguments
215      * passed in from the command line
216      */
217     public static void main(final String ... args) {
218         final String output = new Prog1c(args).toString();
219         System.out.println(output);
220     }
221 }
```

Notes and Instructor Comments