

```
printf \n\n\n      #print three blank lines
```

```
cat p4.sh          #display the shell script file for the program
#!/bin/bash
```

```
set -v            #turn on echo
printf \n\n\n      #print three blank lines
cat p4.sh          #display the shell script file for the program
printf \f          #issue a form feed (top of a new page)
cat -b p4.java     #display the source file with line numbers
:                 #null command
:
:
:
```

```
javac p4.java      #compile the java file
java p4            #execute the file from the current directory
:
:
:
:
```

```
date              #print the date
printf \f          #issue a form feed (top of a new page)
```

```
cat -b p4.java     #display the source file with line numbers
```

```
1  /*
2  PROGRAM NAME: Program 4, Stacks
3  PROGRAMMER:  Samuel Jentsch
4  CLASS:      CSC 241.001, Fall 2013
5  INSTRUCTOR: Dr. D. Dunn
6  DATE STARTED: October 8, 2013
7  DUE DATE:   October 14, 2013
8  REFERENCES: Computer Science
9              Data Abstraction and Problem Solving with
10             Java
11             Frank M. Carrano & Janet J. Prichard
12             Dr. Dunn: assignment information sheet

13 PROGRAM PURPOSE:
14 This program reads a series of commands from a file, parses
15 the commands, and manipulates stacks by adding, traversing,
16 and removing items.

17 VARIABLE DICTIONARY:
18 garage - Stack, used to hold the stack data structure for the
19           program. Switch the initialization to change the
20           implementation.
21
22 ADTs:
23 Stack
```

```

24 FILES USED:
25     p4.dat - a file containing commands and license plates.
26     -----
27 */

28 import java.io.*;
29 import java.util.*;

30 public class p4 {
31     //Change class to test different implementation
32     static Stack garage;
33
34     public static void main(String[] args) {
35         //Change to new StackP() to switch implementations.
36         garage = new StackP();
37
38         File file = new File("../instr/p4.dat");

39         handleInput(file);
40
41         System.out.println("Exiting...");
42         vCommand();
43         System.exit(0);
44     }
45
46     public static void handleInput(File inputFile) {
47         //-----
48         //Takes a file as parameter and reads input line by line
49         //until a 'c' or end of file is reached. Passes input
50         //to parseCommand() to be parsed and run.
51         //Precondition: File passed as parameter containing
52         //commands and license plates.
53         //Postcondition: Passes input line by line to
54         //parseCommand() to be run by the program.
55         //-----
56
57         try {
58             Scanner fileReader = new Scanner(inputFile);
59             boolean continueInput = true;
60             while (fileReader.hasNextLine() && continueInput) {
61                 continueInput = parseCommand(fileReader.nextLine());
62             }
63         } catch (FileNotFoundException e) {
64             // TODO Auto-generated catch block
65             e.printStackTrace();
66         }
67
68     }

```

```

69
70     public static boolean parseCommand(String command) {
71         //-----
72         //Takes a string as a parameter, splits it into a command
73         //character and a license plate (if necessary), and calls
74         //the method corresponding to the command.
75         //Precondition: String input with a single character
76         //command as first letter.
77         //Postcondition: Calls command appropriate to first
78         //character.
79         //-----

80         boolean continueInput = true;
81
82         char commandChar = Character.toUpperCase(command.charAt(0));
83
84         String licensePlate = "";
85
86         if(command.length() > 1)
87             licensePlate = command.substring(1);
88
89         switch (commandChar) {
90             case 'A':
91                 aCommand(licensePlate);
92                 break;
93             case 'D':
94                 dCommand(licensePlate);
95                 break;
96             case 'V':
97                 vCommand();
98                 break;
99             case 'C':
100                 continueInput = false;
101                 break;
102             default:
103                 System.out.println("Unsupported command: " + commandChar);
104         } //switch
105
106         return continueInput;
107     }
108
109     public static boolean validatePlate(String plate) {
110         //-----
111         //Takes a string as a parameter and checks to see if
112         //it is a valid license plate. Valid plates contain
113         //three capital letters followed by three digits.
114         //Precondition: String input plate passed as parameter.
115         //Postcondition: Returns true if plate is a valid plate
116         //String, false if not.

```

```

117 //-----
118
119 boolean isGood = true;
120 char[] plateCharacters = plate.toCharArray();
121 if(plateCharacters.length > 6) {
122     //Doesn't meet length requirement.
123     isGood = false;
124 }
125 else {
126     for(int i = 0; i < plateCharacters.length && isGood == true; i++) {
127         if(i <= 2 && !Character.isLetter(plateCharacters[i])) {
128             //First three characters must be letters.
129             isGood = false;
130         } else if(i > 2 && !Character.isDigit(plateCharacters[i])) {
131             //Last three characters must be digits.
132             isGood = false;
133         } else if(Character.isLetter(plateCharacters[i]) &&
134             !Character.isUpperCase(plateCharacters[i])) {
135             //Letters must be uppercase.
136             isGood = false;
137         }
138     } //end for
139 } //end else
140
141 return isGood;
142 }
143
144 public static void aCommand(String licensePlate) {
145     //-----
146     //Handles car arrival for string licensePlate. Adds car
147     //to stack if license plate is valid, the car isn't in
148     //the garage, and the garage is not full.
149     //Precondition: String input licensePlate.
150     //Postcondition: Adds car
151     //to stack if license plate is valid, the car isn't in
152     //the garage, and the garage is not full. Prints out
153     //appropriate message otherwise.
154     //-----
155
156     if(garage.isFull()) {
157         System.out.println("Couldn't add car: " + licensePlate + ". Garage is full.");
158     } else if(isInGarage(licensePlate)) {
159         System.out.println(licensePlate + " is already in the garage.");
160     } else if(validatePlate(licensePlate)) {
161         System.out.println(licensePlate + " was added to the garage");
162         garage.push(licensePlate);
163     } else
164         System.out.println("Invalid license plate: " + licensePlate);
165 }

```

```

166
167 public static boolean isInGarage(String plate) {
168     //-----
169     //Checks stack to see if car with matching license plate
170     //is in stack.
171     //Precondition: String input plate.
172     //Postcondition: returns true if plate is in stack, false
173     //if not.
174     //-----
175
176     if(!garage.isEmpty()) {
177         String item = garage.pop();
178         boolean found = isInGarage(plate);
179         if(plate.matches(item))
180             found = true;
181         garage.push(item);
182         return found;
183     }
184
185     return false;
186 }
187
188 public static void dCommand(String licensePlate) {
189     //-----
190     //Takes a string as a parameter in license plate form.
191     //Checks to see if it's a valid license plate and if the
192     //stack contains items. If both are true,
193     //deleteCarFromStack() is called, and a message is
194     //printed to console if the car was found or not.
195     //Precondition: String input licensePlate.
196     //Postcondition: Validates license plate and stack,
197     //calls deleteCarFromStack() to delete the car and
198     //prints if the delete operation was successful.
199     //-----
200
201     if(!validatePlate(licensePlate))
202         System.out.println("Invalid license plate: " + licensePlate);
203     else if(garage.isEmpty())
204         System.out.println("Cannot delete a car. The garage is empty.");
205     else {
206         if(deleteCarFromStack(licensePlate))
207             System.out.println(licensePlate + " is departing the garage.");
208         else
209             System.out.println("Could not find car with plate: " + licensePlate);
210     }
211 }
212
213 public static boolean deleteCarFromStack(String plate) {
214     //-----

```

```

215 //Takes a string as a parameter, checks the stack by
216 //popping objects off the stack. If the string does not
217 //match the plate string passed, it is pushed back onto
218 //the stack. The base case for the recursive method is
219 //the stack being empty. A boolean is returned indicating
220 //if the delete was successful.
221 //Precondition: String input plate.
222 //Postcondition: Deletes the matching string from the
223 //stack and returns a boolean if the delete succeeded.
224 //-----
225
226 if(!garage.isEmpty()) {
227     String item = garage.pop();
228     boolean found = deleteCarFromStack(plate);
229     if(!plate.matches(item))
230         garage.push(item);
231     else
232         found = true;
233
234     return found;
235 }
236
237 return false;
238 }
239
240 public static void vCommand() {
241     //-----
242     //Calls printStack if the garage stack is not empty to
243     //print all of the cars in the stack.
244     //Precondition: Class variable garage implementing Stack.
245     //printStack to traverse and print the stack.
246     //Postcondition: Stack is traversed and printed. If the
247     //stack is empty a message is printed.
248     //-----
249
250     if(!garage.isEmpty()) {
251         System.out.println("Cars in garage: ");
252         printStack();
253     } else
254         System.out.println("The garage is empty.");
255 }
256
257 public static void printStack() {
258     //-----
259     //Traverses the garage stack and prints each item.
260     //Base case is empty stack.
261     //Precondition: Class variable garage implementing Stack.
262     //Postcondition: Each item in the stack is printed.
263     //-----

```

```

264         if(!garage.isEmpty()) {
265             String item = garage.pop();
266             System.out.println(" " + item);
267             printStack();
268             garage.push(item);
269         }
270     }
271
272 }//end class

273 interface Stack {
274     //---------------------
275     //Interface for the ADT stack containing the operations.
276     //---------------------
277
278     public void createStack();
279
280     public boolean isEmpty();
281
282     public boolean isFull();
283
284     public void push(String newItem);
285
286     public String pop();
287
288     public void popAll();
289
290     public Object peek();
291
292 }

293 class StackA implements Stack{
294     //Array based implementation.
295     final int MAX_STACK = 10;
296     private String[] items;
297     private int top;
298
299     public StackA() {
300         createStack();
301     }
302
303     public void createStack() {
304         //Creates a new empty stack.
305         items = new String[MAX_STACK];
306         top = -1;
307     }
308
309     public boolean isEmpty() {
310         //Returns true if the stack is empty,

```

```

311         //false otherwise.
312         return top < 0;
313     }
314
315     public boolean isFull() {
316         //returns true if the stack is full, false
317         //otherwise.
318         return top == MAX_STACK - 1;
319     }
320
321     public void push(String newItem) {
322         //Adds newItem to the top of the stack.
323         if(!isFull()) {
324             items[++top] = newItem;
325         }
326     }
327
328     public String pop() {
329         //Retrieves and then removes the top of the stack (the
330         //item that was added most recently).
331         if(!isEmpty()) {
332             return items[top--];
333             //return items[top]; top-- should work too.
334         }
335
336         return null;
337     }
338
339     public void popAll() {
340         //Removes all items from the stack.
341         items = new String[MAX_STACK];
342         top = -1;
343     }
344
345     public Object peek() {
346         //Retrieves the top of the stack. That is, peek
347         //retrieves the item that was added most recently.
348         //Retrieval does not change the stack.
349         if(!isEmpty()) {
350             return items[top];
351         }
352
353         return null;
354     }
355 }

```

```

356 class StackP implements Stack {
357     //Reference based implementation

```



```

358     private Node top;
359     private int count = 0;
360     final int MAX_STACK = 10;
361
362     public StackP() {
363         //Creates an empty stack.
364         createStack();
365     }
366
367     public void createStack() {
368         //Creates a new, empty stack.
369         top = null;
370     }
371
372     public boolean isEmpty() {
373         //Return true if stack is empty,
374         //false if not.
375         return top == null;
376     }
377
378     public boolean isFull() {
379         //returns true if the stack is full, false
380         //otherwise.
381         return count == MAX_STACK;
382     }
383
384     public void push(String newItem) {
385         //Adds newItem to the top of the stack.
386         top = new Node(newItem, top);
387         count++;
388     }
389
390     public String pop() {
391         //Retrieves and then removes the top of the stack (the
392         //item that was added most recently).
393         if(!isEmpty()) {
394             Node temp = top;
395             top = top.next;
396             count--;
397             return temp.plate;
398         }
399
400         return null; //remove
401     }
402
403     public void popAll() {
404         //Removes all items from the stack.
405         top = null;
406         count = 0;

```

```

407     }
408
409     public Object peek() {
410         //Retrieves the top of the stack. That is, peek
411         //retrieves the item that was added most recently.
412         //Retrieval does not change the stack.
413
414         if(!isEmpty()) {
415             return top.plate;
416         }
417
418         return null;//remove

```

```

419 class Node {
420     String plate;
421     Node next;
422
423     public Node() {
424         next = null;
425         plate = "";
426     }
427
428     public Node(String licensePlate, Node next) {
429         this.next = next;
430         this.plate = licensePlate;
431     }
432 }

```

```

:         #null command
:
:
:

```

```

javac p4.java      #compile the java file
java p4           #execute the file from the current directory

```

Cannot delete a car. The garage is empty.

AAA111 was added to the garage

BBB222 was added to the garage

CCC333 was added to the garage

DDD444 was added to the garage

EEE555 was added to the garage

FFF666 was added to the garage

Cars in garage:

FFF666

EEE555

DDD444

CCC333

BBB222

AAA111

GGG777 was added to the garage

HHH888 was added to the garage
Unsupported command: X
KKK999 was added to the garage
LLL000 was added to the garage
Couldn't add car: MMM111. Garage is full.
CCC333 is departing the garage.
LLL111 was added to the garage
Could not find car with plate: CCC333
Couldn't add car: MMM222. Garage is full.
Cars in garage:

LLL111
LLL000
KKK999
HHH888
GGG777
FFF666
EEE555
DDD444
BBB222
AAA111

Invalid license plate: M99988

Exiting...

Cars in garage:

LLL111
LLL000
KKK999
HHH888
GGG777
FFF666
EEE555
DDD444
BBB222
AAA111

:
:
:
:

date

#print the date