```
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
```

```
FILES USED:
  p4.dat - a file containing commands and license plates.
-------------------------------------------------------------------
*/

import java.io.*;
import java.util.*;

public class p4 {
        //Change class to test different implementation
        static Stack garage;

        public static void main(String[] args) {
                //Change to new StackP() to switch implementations.
                garage = new StackP();

                File file = new File("../instr/p4.dat");

                handleInput(file);

                System.out.println("Exiting...");
                vCommand();
                System.exit(0);
        }

        public static void handleInput(File inputFile) {
                //------------------------------------------------------
                //Takes a file as parameter and reads input line by line
                //until a 'c' or end of file is reached. Passes input
                //to parseCommand() to be parsed and run.
                //Precondition: File passed as parameter containing
                //commands and license plates.
                //Postcondition: Passes input line by line to
                //parseCommand() to be run by the program.
                //------------------------------------------------------

                try {
                        Scanner fileReader = new Scanner(inputFile);
                        boolean continueInput = true;
                        while (fileReader.hasNextLine() && continueInput) {
                                continueInput = parseCommand(fileReader.nextLine());
                        }
                } catch (FileNotFoundException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }
```

```

	public static boolean parseCommand(String command) {
		//-------------------------------------------------------
		//Takes a string as a parameter, splits it into a command
		//character and a license plate (if necessary), and calls
		//the method corresponding to the command.
		//Precondition: String input with a single character
		//command as first letter.
		//Postcondition: Calls command appropriate to first
		//character.
		//-------------------------------------------------------

		boolean continueInput = true;

		char commandChar = Character.toUpperCase(command.charAt(0));

		String licensePlate = "";

		if(command.length() > 1)
			licensePlate = command.substring(1);

		switch (commandChar) {
		case 'A':
			aCommand(licensePlate);
			break;
		case 'D':
			dCommand(licensePlate);
			break;
		case 'V':
			vCommand();
			break;
		case 'C':
			continueInput = false;
			break;
		default:
			System.out.println("Unsupported command: "  + commandChar);
		}//switch

		return continueInput;
	}

	public static boolean validatePlate(String plate) {
		//-------------------------------------------------------
		//Takes a string as a parameter and checks to see if
		//it is a valid license plate. Valid plates contain
		//three capital letters followed by three digits.
		//Precondition: String input plate passed as parameter.
		//Postcondition: Returns true if plate is a valid plate
		//String, false if not.
		//-----------------------------------------------------

		boolean isGood = true;
		char[] plateCharacters = plate.toCharArray();
		if(plateCharacters.length > 6) {
			//Doesn't meet length requirement.
			isGood = false;
		}
		else {
			for(int i = 0; i < plateCharacters.length && isGood == true; i++) {
				if(i <= 2 && !Character.isLetter(plateCharacters[i])) {
					//First three characters must be letters.
					isGood = false;
				} else if(i > 2 && !Character.isDigit(plateCharacters[i])) {
					//Last three characters must be digits.
					isGood = false;
				} else if(Character.isLetter(plateCharacters[i]) &&
						!Character.isUpperCase(plateCharacters[i])) {
					//Letters must be uppercase.
					isGood = false;
				}
			}//end for
		}//end else

		return isGood;
	}

	public static void aCommand(String licensePlate) {
		//-----------------------------------------------------
		//Handles car arrival for string licensePlate. Adds car
		//to stack if license plate is valid, the car isn't in
		//the garage, and the garage is not full.
		//Precondition: String input licensePlate.
		//Postcondition: Adds car
		//to stack if license plate is valid, the car isn't in
		//the garage, and the garage is not full. Prints out
		//appropriate message otherwise.
		//-----------------------------------------------------

		if(garage.isFull()) {
			System.out.println("Couldn't add car: " + licensePlate + ". Garage is full.");
		} else if(isInGarage(licensePlate)) {
			System.out.println(licensePlate + " is already in the garage.");
		} else if(validatePlate(licensePlate)) {
			System.out.println(licensePlate + " was added to the garage");
			garage.push(licensePlate);
		} else
			System.out.println("Invalid license plate: " + licensePlate);
	}

	public static boolean isInGarage(String plate) {
		//-----------------------------------------------------
		//Checks stack to see if car with matching license plate
		//is in stack.
		//Precondition: String input plate.
		//Postcondition: returns true if plate is in stack, false
		//if not.
		//-----------------------------------------------------

		if(!garage.isEmpty()) {
			String item = garage.pop();
			boolean found = isInGarage(plate);
			if(plate.matches(item))
				found = true;
			garage.push(item);
			return found;
		}

		return false;
	}

	public static void dCommand(String licensePlate) {
		//-----------------------------------------------------
		//Takes a string as a parameter in license plate form.
		//Checks to see if it's a valid license plate and if the
		//stack contains items. If both are true,
		//deleteCarFromStack() is called, and a message is
		//printed to console if the car was found or not.
		//Precondition: String input licensePlate.
		//Postcondition: Validates license plate and stack,
		//calls deleteCarFromStack() to delete the car and
		//prints if the delete operation was successful.
		//-----------------------------------------------------

		if(!validatePlate(licensePlate))
			System.out.println("Invalid license plate: " + licensePlate);
		else if(garage.isEmpty())
			System.out.println("Cannot delete a car. The garage is empty.");
		else {
			if(deleteCarFromStack(licensePlate))
				System.out.println(licensePlate + " is departing the garage.");
			else
				System.out.println("Could not find car with plate: " + licensePlate);
		}
	}

	public static boolean deleteCarFromStack(String plate) {
		//-----------------------------------------------------
		//Takes a string as a parameter, checks the stack by
		//popping objects off the stack. If the string does not
		//match the plate string passed, it is pushed back onto
		//the stack. The base case for the recursive method is
		//the stack being empty. A boolean is returned indicating
		//if the delete was successful.
		//Precondition: String input plate.
		//Postcondition: Deletes the matching string from the
		//stack and returns a boolean if the delete succeeded.
		//-----------------------------------------------------

		if(!garage.isEmpty()) {
			String item = garage.pop();
			boolean found = deleteCarFromStack(plate);
			if(!plate.matches(item))
				garage.push(item);
			else
				found = true;

			return found;
		}

		return false;
	}

	public static void vCommand() {
		//-----------------------------------------------------
		//Calls printStack if the garage stack is not empty to
		//print all of the cars in the stack.
		//Precondition: Class variable garage implementing Stack.
		//printStack to traverse and print the stack.
		//Postcondition: Stack is traversed and printed. If the
		//stack is empty a message is printed.
		//-----------------------------------------------------

		if(!garage.isEmpty()) {
			System.out.println("Cars in garage: ");
			printStack();
		} else
			System.out.println("The garage is empty.");
	}

	public static void printStack() {
		//-----------------------------------------------------
		//Traverses the garage stack and prints each item.
		//Base case is empty stack.
		//Precondition: Class variable garage implementing Stack.
		//Postcondition: Each item in the stack is printed.
		//-----------------------------------------------------
```

```
			if(!garage.isEmpty()) {
				String item = garage.pop();
				System.out.println("  " + item);
				printStack();
				garage.push(item);
			}
		}

}//end class

interface Stack {
		//----------------------------------------------------
		//Interface for the ADT stack containing the operations.
		//----------------------------------------------------

		public void createStack();

		public boolean isEmpty();

		public boolean isFull();

		public void push(String newItem);

		public String pop();

		public void popAll();

		public Object peek();

}

class StackA implements Stack{
		//Array based implementation.
		final int MAX_STACK = 10;
		private String[] items;
		private int top;

		public StackA() {
			createStack();
		}

		public void createStack() {
			//Creates a new empty stack.
			items = new String[MAX_STACK];
			top = -1;
		}

		public boolean isEmpty() {
			//Returns true if the stack is empty,
			//false otherwise.
			return top < 0;
		}

		public boolean isFull() {
			//returns true if the stack is full, false
			//otherwise.
			return top == MAX_STACK - 1;
		}

		public void push(String newItem) {
			//Adds newItem to the top of the stack.
			if(!isFull()) {
				items[++top] = newItem;
			}
		}

		public String pop() {
			//Retrieves and then removes the top of the stack (the
			//item that was added most recently).
			if(!isEmpty()) {
				return items[top--];
				//return items[top]; top-- should work too.
			}

			return null;
		}

		public void popAll() {
			//Removes all items from the stack.
			items = new String[MAX_STACK];
			top = -1;
		}

		public Object peek() {
			//Retrieves the top of the stack. That is, peek
			//retrieves the item that was added most recently.
			//Retrieval does not change the stack.
			if(!isEmpty()) {
				return items[top];
			}

			return null;
		}

}

class StackP implements Stack {
		//Reference based implementation
```

```
	private Node top;
	private int count = 0;
	final int MAX_STACK = 10;

	public StackP() {
		//Creates an empty stack.
		createStack();
	}

	public void createStack() {
		//Creates a new, empty stack.
		top = null;
	}

	public boolean isEmpty() {
		//Return true if stack is empty,
		//false if not.
		return top == null;
	}

	public boolean isFull() {
		//returns true if the stack is full, false
		//otherwise.
		return count == MAX_STACK;
	}

	public void push(String newItem) {
		//Adds newItem to the top of the stack.
		top = new Node(newItem, top);
		count++;
	}

	public String pop() {
		//Retrieves and then removes the top of the stack (the
		//item that was added most recently).
		if(!isEmpty()) {
			Node temp = top;
			top = top.next;
			count--;
			return temp.plate;
		}

		return null; //remove
	}

	public void popAll() {
		//Removes all items from the stack.
		top = null;
		count = 0;
```

```
    }

    public Object peek() {
        //Retrieves the top of the stack. That is, peek
        //retrieves the item that was added most recently.
        //Retrieval does not change the stack.

        if(!isEmpty()) {
            return top.plate;
        }

        return null;//remove
    }
}

class Node {
    String plate;
    Node next;

    public Node() {
        next = null;
        plate = "";
    }

    public Node(String licensePlate, Node next) {
        this.next = next;
        this.plate = licensePlate;
    }
}
```

```
:                  #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
```