University of Washington Bothell
Computing and Software Systems
Graduate Certificate Program
Programming Competency Self-Assessment: First Assessment (of two)

There are two program specifications in this self-assessment. Candidates to the CSS Graduate Certificate Program should find the completion of both of the programs to be *straightforward* and *easy*. Candidates who find the following program development to be challenging should consider taking CSS161.

Program Specification One (of two):

You work at a soft drink distributorship that sells at most 100 different kinds of soft drinks. The program you write for this assignment will process weekly transactions and allow for a report to be displayed that includes the soft-drink name, ID, starting inventory, final inventory, and the number of transactions received.

You should write two data files, data6.txt and data6trans.txt, which should hold the initial soft drink information and transactions, respectively. The file data6.txt should consist of at most 100 lines where each line contains the soft drink name (one string), ID (string), and the starting inventory of cases (int).

The file data6trans.txt should hold the transactions. Each transaction should consist of the ID followed by the number of cases purchased (positive integer), or the amount sold (negative integer). You can assume the format of the data is correct, but not all IDs are valid. In the case of an invalid ID, do not process the data (ignore it, no error message).

Sample data6.txt:

```
Coke       123 100
Pepsi      345 50
CanadaDry  678 75
DrPepper   444 120
```

Sample data6trans.txt:

```
345   10
123   -5
345   10
678    8
444   20
444  -20
444   10
999    5
345   10
123  -25
```

The displayReport function displays the drink name, ID, starting inventory, final inventory, and the number of transactions processed. Display this exact format (number of blanks separating items does not have to be identical).

For the sample data, the output of your program would be as follows:

```
Soft drink      ID     Starting Inventory    Final Inventory   # transactions
Coke            123          100                    70                2
Pepsi           345           50                    80                3
CanadaDry       678           75                    83                1
DrPepper        444          120                   130                3
```

Write a SoftDrinkInventory class with the following functionality:

`public constructor`: Initializes arrays holding soft drink name and ID to hold all empty strings (calls intitializeString twice to perform the tasks). Initializes arrays holding starting inventory, final inventory, and the counts of the number of transaction to zero (calls initializeInt three times to perform the tasks).

`public buildInventory`: Sets the arrays for soft drink name, ID, and starting inventory from information in the data file. The array holding final inventory is set to the same values as the starting inventory.

`public processTransactions`: Processes the transactions by correctly adjusting the final inventory and transaction counts arrays. Data for IDs which don't exist are not processed.

`public displayReport`: Displays a report including soft drink name, ID, starting inventory, final inventory, and number of transactions processed.

`private findID`: Takes an ID parameter and returns the position in the array (the subscript) where the soft drink with that ID is found. Return -1 if the ID is not found.

`private initializeInt`: Takes an int array parameter and initializes all array values to zero.

`private initializeString`: Takes a String array parameter and initializes all values to the empty String (“”).

```java
/**
 * This program tests the functionality of the SoftDrinkInventory class.
 * An object is constructed which initially holds no real data.
 * A datafile containing initial data is used to fill the SoftDrink object.
 * Then transactions are processed where each transaction contains how
 * cases are bought or sold. A function displays a report of the drink name,
 * ID, starting inventory, final inventory, and the number of transactions
 * processed.
 */
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class SoftDrinkTester {

    public static void main (String[] args) {
        Scanner inventoryFile = null;              // inventory data file
        Scanner transFile = null;                  // transaction data file

        // open the inventory initialization file
        try {
            inventoryFile = new Scanner(new FileInputStream("data6.txt"));
        }
        catch (FileNotFoundException e) {
            System.out.println("File not found or not opened.");
            System.exit(0);
        }

        // open the file containing the buy/sell transactions
        try {
            transFile = new Scanner(new FileInputStream("data6trans.txt"));
        }
        catch (FileNotFoundException e) {
            System.out.println("File not found or not opened.");
            System.exit(0);
        }

        // instantiate the soft drink distributorship object
        // and process the transactions by updating the inventory totals
        SoftDrinkInventory softDrinks = new SoftDrinkInventory();
        softDrinks.buildInventory(inventoryFile);
        softDrinks.processTransactions(transFile);
        softDrinks.displayReport();
    }
}
```

Program Specification Two (of two):

This is the hangperson (formerly known as hangman) game where a player tries to guess a word given the spaces it takes up. Each time an incorrect letter is chosen, another body part is added to the gallows. After seven tries, the whole body is hung and the player loses.

Before the user plays, an introduction is displayed briefly describing the game. At the start of one round of the game, empty gallows are display, dashes or underscores appear where the letters would be, and where the guessed letters will be displayed. Additionally letters that the user has already guessed are displayed. As the user plays the game, the gallows, the word to guess, the letters guessed are continually updated. After the player wins or loses one round, a prompt asks if the player wants to continue playing. Call your class HangPerson. The class with main and the high level pseudocode for the function *play()* are found below.

You can use any display besides a hanging to demonstrate missing letters, e.g., flowers turning into weeds, or a car crashing against a wall. The only characteristic that must remain is that a player only gets seven guesses. Below are two sample runs of the program.

```
Welcome to the hangperson game ...
To play, guess a letter to try to guess the word.
Every time you choose an incorrect letter another
body part appears on the gallows. If you guess the
word before you're hung, you win :-)
If you get hung you lose :-(

Time to guess ...

|-----|-
|
|
|
|
|__________

Letters guessed already =>
Number of wrong guesses => 0
The word so far =>  -----

Choose a letter => e

|-----|-
|
|
|
|
|__________

Letters guessed already => e
Number of wrong guesses => 0
The word so far =>  -e---

Choose a letter => x

|-----|-
|     O
|
|
|
|__________

Letters guessed already => e x
Number of wrong guesses => 1
The word so far =>  -e---

Choose a letter => 3

Invalid, don't you know your ABCs?
```

```
Choose a letter => l

|-----|-
|     O
|
|
|
|__________

Letters guessed already => e l x
Number of wrong guesses => 1
The word so far =>  -ell-

Choose a letter => e


You already tried this letter

Choose a letter => l


You already tried this letter

Choose a letter => o

|-----|-
|     O
|
|
|
|__________

Letters guessed already => e l o x
Number of wrong guesses => 1
The word so far =>  -ello

Choose a letter => h

|-----|-
|     O
|
|
|
|__________

Letters guessed already => e h l o x
Number of wrong guesses => 1
The word so far =>  hello
```

```
Congratulations, you win!!!
Do you want to play again? yes
```

```
|-----|-
|
|
|
|
|__________

Letters guessed already =>
Number of wrong guesses => 0
The word so far =>  -----

Choose a letter => x

|-----|-
|     O
|
|
|
|__________

Letters guessed already => x
Number of wrong guesses => 1
The word so far =>  -----

Choose a letter => z

|-----|-
|     O
|     |
|
|
|__________

Letters guessed already => x z
Number of wrong guesses => 2
The word so far =>  -----

Choose a letter => t

|-----|-
|     O
|     |
|     |
|
|__________

Letters guessed already => t x z
Number of wrong guesses => 3
The word so far =>  -----

Choose a letter => p

|-----|-
|     O
|    \|
|     |
|
|__________
```

```
Letters guessed already => p t x z
Number of wrong guesses => 4
The word so far =>  -----

Choose a letter => y

|-----|-
|     O
|    \|/
|     |
|
|__________

Letters guessed already => p t x y z
Number of wrong guesses => 5
The word so far =>  -----

Choose a letter => m

|-----|-
|     O
|    \|/
|     |
|    /
|__________

Letters guessed already => m p t x y z
Number of wrong guesses => 6
The word so far =>  -----

Choose a letter => n

|-----|-
|     O
|    \|/
|     |
|    / \
|__________

Letters guessed already => m n p t x y z
Number of wrong guesses => 7
The word so far =>  -----

Too bad, you lose!
The word was ==> world
Do you want to play again? no


Thanks for playing!
```

After reading a String from the datafile, it (and anything related) is stored as arrays of char. The function `toCharArray` converts a string to a char array. You must use a boolean array to keep track of whether or not a letter was guessed. The integers 0 to 25 are mapped to the letters ‘a’ to ‘z’. For example, suppose letter was ‘e’, then `letter-'a'` is 5, the difference between the internal value of 'e' and 'a' . This is how you get to subscript 5 for an 'e'.

The class HangPersonGames

```java
/* This program is a word guessing game called hangperson.
 * A person will try and guess a word before the max
 * number of guesses are used. Every time the user chooses an
 * incorrect letter another body part is displayed in the gallows.
 */

import java.util.*;
import java.io.FileInputStream;
import java.io.FileNotFoundException;

public class HangPersonGames {

    public static void main(String[] args) {
        Scanner wordsFile = null;                  // words data file

        // open the file containing the words to guess
        try {
            wordsFile = new Scanner(new FileInputStream("data7.txt"));
        }
        catch (FileNotFoundException e) {
            System.out.println("File not found or not opened.");
            System.exit(0);
        }

        HangPerson hangGame = new HangPerson(wordsFile);
        Scanner keyboard = new Scanner(System.in);

        // display an introduction on the game to the player
        hangGame.displayGameIntro();

        // continually play new games if the user desires
        String playAgain;
        do {
            hangGame.play();
            System.out.print("Do you want to play again? ");
            playAgain = keyboard.next();
            System.out.println();
        } while (playAgain.toUpperCase().startsWith("Y"));

        System.out.println();
        System.out.println("Thanks for playing!");
        System.out.println();
    }
}
```

High-level pseudocode for the play() function

```
initialization of everything
get one word from the datafile
loop - exit when you win or lose
    display including gallows, letters guessed, wrong guesses, word so far
    get a valid, never-guessed-before letter from the user
    decide if the letter is in the word and do all appropriate updating
end loop
handle winning or losing
```