CS 220 Software Design II — Spring 2022

Midterm Practice Problems

University of Wisconsin - La Crosse — *March 8*

Consider the following classes for the next two problems.

```
public class Engine {
    protected int numCylinders;
    protected int hp;
    public Engine(int c, int hp)
    {
        numCylinders = c;
        this.hp = hp;
    }
    public String drive() {
        if(numCylinders >= 8)
            return "VROOM";
        else
            return "vroom";
    }
    public void tune(int p) {
         hp += p;
    }
    public int getHP() {
        return hp;
    }
}
```

```
public class Supercharged extends Engine {
    protected int boost;
    public Supercharged(int n, int hp, int
    p) {
        super(n, hp);
        boost = p;
    }
    public String drive() {
        String str = "Whine";
        for (int i = 0; i < boost/3; ++i) {
            str += "!";
        }
        return str;
    }
    public int funFactor() {
        return boost * 10;
    }
}
```

1. What is printed to the console as a result of this code fragment?

```
1 Engine e = new Supercharged(4, 200, 17);
2 System.out.println(e.getHP() + ": " + e.drive());
3 e.tune(10);
4 System.out.println(e.getHP() + ": " + e.drive());
```

Solution:

```
200: Whine!!!!!
210: Whine!!!!!
```

2. Cross out the lines of code that are syntactically invalid. In the space below, for each line you cross out, note the line number and **why** that line is invalid.

```
1 Supercharged e2 = new Supercharged(4, 210, 12);
2 Engine e3 = new Supercharged(8, 600, 20);
3 e3.tune();
4 System.out.println(e2.funFactor());
5 System.out.println(e3.funFactor());
```

Solution:

```
5: method is not available for e3
```

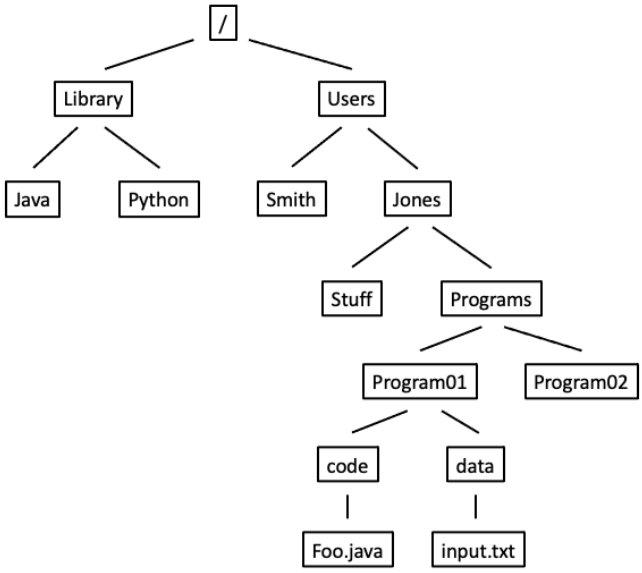
3. Write a new interface `WinChampionsLeague` that has three methods: a method `buyPlayers` that takes a single `int` parameter representing the number of players to buy and returns a `int` representing the amount spent, a method `fireManager`, that takes a `String` parameter representing the name of the new manager, and a method `formation` that takes three `int` parameters representing the number of defenders, number of midfielders and the number of attacking players in the new lineup, and returns a boolean indicating whether the number of players totals 10.

Solution:

```
public interface WinChampionsLeague {

    public int buyPlayers(int numPlayers);
    public void fireManager(String newManager);
    public boolean formation(int d, int m, int a);
}
```

Consider the file system below for the following problems:



4. How can you refer to the file `input.txt` with an absolute path?

Solution:
`/Users/Jones/Programs/Program01/data/input.txt`

5. How can you refer to the file `input.txt` with a relative path within `Foo.java`?

Solution:
`../data/input.txt`

6. Write a code fragment to read in the lines contained in the **text** file `lyric.txt` (do **not** use `Scanner` to read from the input file). Write out the number of characters in each `String` (in the same order, one on each line) to the file `counts.txt`. Note that `counts.txt` might already contain data - **do not overwrite it**. An example of an input and corresponding output file appears below:

`lyric.txt`

```
Every time I get the inspiration
To go change things around
No one wants to help look for places
Where new things might be found
Where can I turn when my fair weather friends cop out
What's it all about
```

`counts.txt`

```
32
26
36
31
53
19
```

Solution:

Text File Solution

```
1  FileReader fr = new FileReader(new File("lyric.txt"));
2  BufferedReader br = new BufferedReader(fr);
3
4  FileWriter fw = new FileWriter(new File("counts.txt", true));
5  BufferedWriter bw = new BufferedWriter(fw);
6  PrintWriter pw = new PrintWriter(bw);
7
8  String line = br.readLine();
9  while(line != null) {
10     int len = line.length();
11     pw.println(len);
12
13     line = br.readLine();
14 }
15
16 br.close();
17 pw.close();
```

7. Write a `main` method that reads `int` values from a text file and creates a 2D array that contains the data. The format of the file is: the first line contains an `int` value that represents the number of data lines that follow. The remainder of the file is a sequence of lines, each of which contains all of the `int` values for a row of the 2D array. Within a line of the file, the values are separated by spaces. You may assume that the number of rows is greater than 0. An example input file appears below:

`input.txt`

```
7
182 2 12 121 135 96 53 13
197 132 143 160 30 161 138 17 15 112 25
127 54 70
173 95 5 156 119 107 96 90
43 102 172 136 162
76 138 86 48 175 81 85 1 25 198 127 29
192 151 174 20 64 111 93 107 38 79
```

Solution:

```
1  public static void main(String[] args) {
2      FileReader fr = new FileReader(new File("lyric.txt"));
3      BufferedReader br = new BufferedReader(fr);
4
5      String line = br.readLine();
6      int rows = (int)line;
7
8      int[][] dataArray = new int[rows][];
9
10     int i = 0;
11     while(i < rows) {
12         line = br.readLine();
13         Scanner scan = new Scanner(line);
14         nt count = 0;
15         while(scan.hasNextInt()) {
16             count++;
17             scan.nextInt();
18         }
19         scan.close();
20
21         dataArray[i] = new int[count];
22         Scanner scan = new Scanner(line);
23         int col = 0;
24         while(scan.hasNextInt()) {
25             dataArray[i][col] = scan.nextInt();
26         }
27         scan.close();
28
29         i++;
30     }
31
32     br.close();
33 }
```

8. Consider the partial implementation of the `ArrayList` class below. Implement a `private` method `clear` that removes all data from the `ArrayList` and resets the length of the underlying array to `DEFAULT_CAPACITY`.

```
1  public class ArrayList<E> {
2
3      private static int DEFAULT_CAPACITY = 10;
4      private Object data[];
5      private int size;
6
7      public ArrayList(int index) { ... }
8
9      public void add(E e) { ... }
10
11     public boolean add(int index, E e) { ... }
12
13     public E remove(int index) { ... }
14
15 }
```

Solution:

```
    public ArrayList<E> clear() {

        data = new Object[DEFAULT_CAPACITY];
        size = 0;
    }
```

9. Given the `SinglyLinkedList` class that appears at the end of this document, write the instance method `shuffle` within the `SinglyLinkedList` class. `shuffle` takes a `SinglyLinkedList other` as a parameter. It alters the `SinglyLinkedList` to which the method is applied by interleaving values from the two lists, with each value from `other` being second in each "pair." When the method completes, `other` should be empty. You can assume that the two lists have the same non-zero length.

Solution:

```
public void shuffle(SinglyLinkedList<E> other) {
    int count = other.size;
    int i = 0;
    SingleListNode curNode = firstNode;
    SingleListNode curNodeOther = other.firstNode;
    while(i < count) {
        curNodeOther = curNodeOther.nextNode;
        other.firstNode.nextNode = curNode.nextNode;
        curNode.nextNode = other.firstNode;
        other.firstNode = curNodeOther;
        curNode = curNode.nextNode.nextNode;

        size++;
        i++;
    }
    other.size = 0;
}
```

Reference Classes

```
1  public class SinglyLinkedList<E> {
2      private int size;
3      private SingleListNode firstNode;
4
5      public SinglyLinkedList() {
6          size = 0;
7          // assumes the use of a sentinel node
8          firstNode = new SingleListNode(null);
9      }
10
11     private class SingleListNode {
12         private E data;
13         private SingleListNode nextNode;
14
15         public SingleListNode(E i) {
16             data = i;
17             nextNode = null;
18         }
19     }
20 }
```