Midterm 1: CompSci 201

Prof. Astrachan

October 4, 2023

Name: _____

netid: _____

In submitting this test, I affirm that I have followed the Duke Community Standard.

Community standard acknowledgement (signature) _____

Some questions have two answer possibilities, some have three, some have four. Make sure you bubble answers appropriately.

This Exam is Form C, please mark your answer sheet accordingly

Problems about NBody

In the `CelstialBody` class from P1-NBody there are six *getter* methods: `getMass`, `getName`, `getX`, `getY`, `getXVel`, and `getYVel`. Each of these methods consists of one line that returns the value of an instance variable `myVAR` as in:

```
return myVAR;
```

For exampe, for method `getXVel` the code in the method is `return myXVel`.

All the methods in `CelestialBody` have access to the instance variables directly, without calling the `getVAR` methods. For example, the code below is from a student solution for the method `calcDistance`:

```
106         public double calcDistance(CelestialBody b) {
107                 // TODO: complete method
108                 double dx = myXPos - b.getX(); double dy = myYPos - b.getY();
109                 double r = Math.pow(dx,2) + Math.pow(dy,2);
110                 r = Math.sqrt(r);
111                 return r;
112         }
```

This shows the student using `b.getX()`, for example. However, since `calcDistance` is a method of the `CelestialBody` class, this expression could be written as `b.myXPos` since the code in the `calcDistance` class has access to all instance variables in the class.

PROBLEM 1:

The **only** use of *getter* methods in the `NBody` simulation class is in the print statements shown below.

```
128                 for (int i = 0; i < bodies.length; i++) {
129                     System.out.printf("%11.4e %11.4e %11.4e %11.4e %11.4e %12s\n",
130                                                 bodies[i].getX(), bodies[i].getY(),
131                                         bodies[i].getXVel(), bodies[i].getYVel(),
132                                         bodies[i].getMass(), bodies[i].getName());
133                 }
134         }
```

True or False, if there had been an appropriate `toString` method in the `CelestialBody` class (there is none in the code you wrote) the *getter* methods would not be needed in either the `NBody` class or the `CelestialBody` class since the `toString` method could be called automatically or via code to print the information for a `CelestialBody`.

A. True, the *getter* methods would not be needed in these two classes if there was an appropriate `toString` method.

B. False, the *getter* methods would be needed in one of the two classes.

PROBLEM 2:

True or False, in the code shown above for method `calcDistance`, the statement `double dx = myXPos - b.getX()` can be replaced by `double dx = b.getX() - myXPos` without changing any results – the code is still correct.

A. True, the code is still correct in all uses of `calcDistance`.

B. False, changing the order will yield errors since distance will be in the opposite direction or incorrect.

PROBLEM 3:

In the code you wrote for `CelestialBody` there is only one constructor (you were asked to write it):

```
31          public CelestialBody(double xp, double yp, double xv,
32                                  double yv, double mass, String filename){
33                  // TODO: complete constructor
34
35          }
```

To be correct, your code assigned values to six instance variables, one for each parameter to the constructor. True or False, the order of the assignments must match the order of the parameters, so the first statement in any correct constructor must be:

```
myXPos = xp;
```

You cannot, for example, have the first statement by `myYPos = yp`.

A. True, the first statement must be `myXPos = xp;`

B. False, you can assign values to the six instance variables in any order.

Printing Questions

Consider the four print statements in the code below:

```
48    String s = new String("duke");
49    String t = s;
50    String[] a = {"bat", "dog", "cat", "ant", "fox"};
51    String[] b = a;
52    s = s + "duke";
53    a[1] = a[4];
54    System.out.printf("%s \t %d\n",s, s.length());
55    System.out.printf("%s \t %d\n",t, t.length());
56    System.out.printf("%s \t %s\n",a[1], b[1]);
57    System.out.printf("%s \t %s\n",b[4]+a[4], a[4] + b[4]);
```

PROBLEM 4:

What is printed by the print statement on line 54?

A. duke 4

B. dukeduke 8

C. dukedukeduke 12

PROBLEM 5:

What is printed by the print statement on line 55?

A. duke 4

B. dukeduke 8

C. dukedukeduke 12

PROBLEM 6:

What is printed by the print statement on line 56?

A. duke duke

B. fox fox

C. fox duke

D. duke fox

Hashing Stuff

For each statement, choose between True, False, or cannot be determined from what's given. If you choose *cannot be determined* that means for some values of variables `s` and `t` the statement could be true, but for other values it could be false. If you choose true, then the statement is true for any/all values of `s` and `t`, and similarly if you choose false, the statement is false for any/all values.

PROBLEM 7:

If two string variables s and t have different hash code values, that is `s.hashCode() != t.hashCode()`, then what is the the value of `s.equals(t)`?

A. True

B. False

C. Cannot be determined from the information given

PROBLEM 8:

If two string variables s and t have `s == t` (is true); what is the the value of `s.equals(t)`?

A. True

B. False

C. Cannot be determined from the information given

PROBLEM 9:

If two string variables s and t have `s.equals(t) == false` then what is the value of `s.hashCode() == t.hashCode()`.

A. True

B. False

C. Cannot be determined from the information given

PROBLEM 10:

Which of the following best characterizes the runtime complexity of the call `summer(N)`?

```
public int summer(int num) {
      int sum = 0;
      for(int k=0; k < num/2; k++) {
          sum += k;
      }
      return sum;
  }
```

A. $O(\log N)$

B. $O(\sqrt{N})$

C. $O(N)$

D. $O(N \log N)$

E. $O(N^2)$

PROBLEM 11:

Which of the following best characterizes the *value returned* by the call `summer(N)` (code above)? Note that `summer(10) == 10`.

A. $O(\log N)$

B. $O(\sqrt{N})$

C. $O(N)$

D. $O(N \log N)$

E. $O(N^2)$

PROBLEM 12:

Which of the following best characterizes the runtime complexity of the call `value(N)`?

```
public int value(int num) {
    int sum = 0;
    for(int k=0; k < num; k++) {
        for(int j=1; j <= num; j *= 2) {
            sum += 1;
        }
    }
    return sum;
 }
```

A. $O(\log N)$

B. $O(\sqrt{N})$

C. $O(N)$

D. $O(N \log N)$

E. $O(N^2)$

PROBLEM 13:

Which of the following best characterizes the runtime complexity of the call `calc(N)`?

```
public int calc(int num) {
    int sum = 0;
    int val = 1;
    while (val*val <= num) {
        sum += 1;
        val += 1;
    }
    return sum;
 }
```

A. $O(\log N)$

B. $O(\sqrt{N})$

C. $O(N)$

D. $O(N \log N)$

E. $O(N^2)$

Problems about Arrays

The code below shows the creation of an `ArrayList` object and additions to this object.

```
29      String[] a = {"one", "two", "three"};
30      ArrayList<String> list = new ArrayList<>(Arrays.asList(a));
31      System.out.printf("%s \t %d\n",list,list.size());
32
33      list.addAll(list);
34      System.out.printf("%s \t %d\n",list,list.size());
35
36      list.addAll(list);
37      System.out.printf("%s \t %d\n",list,list.size());
```

PROBLEM 14:

The first print statement on line 31 prints

```
[one, two, three]  3
```

What is printed by the print statement on line 34?

A. `[one, two, three] 3`

B. `[one, two, three, one, two, three] 3`

C. `[one, two, three, one, two, three] 6`

D. Nothing, the code on line 33 throws a *concurrent modification exception*

PROBLEM 15:

Consider code added after the `printf` statement on line 31, i.e., the code below is added where line 32 is shown above.

```
HashSet<String> set = new HashSet<>();
set.addAll(list);
set.addAll(list);
set.addAll(list);
System.out.printf("%s \t %d\n",set,set.size());
```

Adding the three calls of `set.addAll` as shown and then printing the resulting set will result in printing (by the newly added `System.out.printf` statement shown).

A. `[one, two, three] 3`

B. `[one, two, three, one, two, three] 3`

C. `[one, two, three, one, two, three] 6`

D. `[one, two, three, one, two, three, one, two, three] 9`

Markov, Polov

In the *P2: Markov* assignment the same method `getRandomText` below was used in both class `BaseMarkov` and in class `HashMarkov`. Questions about the method follow the code.

```
106     public String getRandomText(int length){
107         ArrayList<String> randomWords = new ArrayList<>(length);
108         int index = myRandom.nextInt(myWords.length - myOrder + 1);
109         WordGram current = new WordGram(myWords,index,myOrder);
110         randomWords.add(current.toString());
111
112         for(int k=0; k < length-myOrder; k += 1) {
113             String nextWord = getNextWord(current);
114             if (nextWord.equals(END_OF_TEXT)) {
115                 break;
116             }
117             randomWords.add(nextWord);
118             current = current.shiftAdd(nextWord);
119         }
120         return String.join(" ", randomWords);
121     }
```

PROBLEM 16:

One line in the code above executes more quickly for `HashMarkov` than for `BaseMarkov`. **What line executes more quickly**?

A. 110

B. 113

C. 117

D. 120

PROBLEM 17:

The call `getRandomText(N)` shown above runs in $O(N)$ time to generate `N` random characters when `HashMarkov` is used. If line 107 is replaced by `String ret = "";` line 117 is replaced by `ret += nextWord+" ";` and line 120 replaced by `return ret.trim()` the output will be exactly the same as in the code above. With this change, what will the runtime be? Assume all strings have fewer than 30 characters.

A. $O(N)$

B. $O(N \log N)$

C. $O(N^2)$

D. $O(N^3)$

Mapping

Consider a `Map<String,String[]> map` in which keys are student names and corresponding values are an array of classes taken by each student. One example of a map is shown below where the keys in the map are *Blake, Taylor, Jessie, and Sam* and the courses for each student are shown.

```
Blake:   [compsci 201, math 218, phil 250]
Taylor:  [econ 101, math 218, bio 201, ics 101]
Jessie:  [compsci 201, bio 201, ics 101, phil 250]
Sam:     [econ 101, compsci 201, bio 201, math 218]
```

PROBLEM 18:

The method `mprint` shown below prints the contents of its parameter in the format shown above. The statement on line 79 is missing from `mprint`, what is that statement so the output above would be generated? Each statement below compiles, but only one generates the output shown above.

```
76      public static void mprint(Map<String,String[]> map) {
77          for(String key : map.keySet()) {
78              System.out.printf("%s:\t", key);
79              // missing statement here
80          }
81      }
```

A. `System.out.println(map.get(key))`

B. `System.out.println(Arrays.toString(map.get(key))`

C. `System.out.println(map.values());`

D. `System.out.println(map.getOrDefault(key,null))`

New Map

The method `createRosters` (next page) returns a `Map<String,String[]>` in which keys are class names and corresponding values are an array of the students in each class. For the map printed above, the map returned by `createRosters(map)` is:

```
bio 201:     [Taylor, Jessie, Sam]
phil 250:    [Blake, Jessie]
math 218:    [Blake, Taylor, Sam]
econ 101:    [Taylor, Sam]
ics 101:     [Taylor, Jessie]
compsci 201: [Blake, Jessie, Sam]
```

The code for `createRosters` is shown below with two statements and one code segment missing:

```
33  public static Map<String, String[]> createRosters(Map<String, String[]> map){
34      HashMap<String, ArrayList<String>> temp = new HashMap<>();
35      for(String key : map.keySet()) {
36          for(String each : map.get(key)) {
37              if (! temp.containsKey(each)) {
38                  // statement 1
39              }
40              // statement 2
41          }
42      }
43      // create map to return
44      Map<String,String[]> ret  = new HashMap<>();
45      // code not shown
46      return ret;
47  }
```

PROBLEM 19:

Which one of the following can replace `statement 1` so that the code works as intended?

A. `temp.put(each, (String[]) map.get(key))`

B. `temp.putIfAbsent(each, String[]{});`

C. `temp.put(each, new ArrayList<>());`

D. `temp.put(each, new ArrayList<>(map.get(key)))`

PROBLEM 20:

Which one of the following can replace `statement 2` so that the code works as intended?

A. `temp.get(each).add(key)`

B. `temp.get(key).add(each)`

C. `temp.putIfAbsent(each,key)`

D. `temp.putIfAbsent(each, new ArrayList<>())`

Duplicating ArrayList

In this section you'll analyze two methods for duplicating a list of strings to create a new list that is *doubled in place*, e.g., so that the list

```
["ape", "bat", "cat", "dog"]
```

could be the source to create a new list as follows

```
["ape", "ape", "bat", "bat", "cat", "cat", "dog", "dog"]
```

Two methods that each return a new list that's *doubled in place* are shown below.

The method `duplicateA` iterates through the list `copy` and adds a duplicate after each list-element using the `ListIterator` methods `hasNext`, `next`, and `add` (we have not covered the interface `ListIterator` in class, you'll reason about it by thinking about the code).

The method `duplicateB` iterates using indexing through the list `copy` setting two values in `copy` so that the returned list is *doubled in place*. Note that `copy` is created so that `copy.size() == 2*list.size()` after line 24

```
4  public static List<String> duplicateA(List<String> list){
5      List<String> copy = new ArrayList<>(list);
6      ListIterator<String> iter = copy.listIterator();
7      while (iter.hasNext()){
8          String s = iter.next();
9          iter.add(s);
10     }
11     return copy;
12 }
```

```
21 public static List<String> duplicateB(List<String> list){
22     int originalSize = list.size();
23     List<String> copy = new ArrayList<>(list);
24     copy.addAll(list);
25     for(int k=originalSize-1; k >= 0; k -= 1){
26         String current = copy.get(k);
27         copy.set(2*k, current);
28         copy.set(2*k+1,current);
29     }
30     return copy;
31 }
```

Answer each of the questions that follow based on these methods.

PROBLEM 21:

Which **one of** the following statements is **false** about `copy` after lines 5 and 23 execute in each of the methods?

A. `copy.get(k).equals(list.get(k))`, for $0 \leq k < list.size()$

B. `copy.size() == list.size()`

C. Executing `copy.set(0,"hello")` would result in `list.get(0).equals("hello")` being true

D. `copy.get(0) == list.get(0)` (note use of `==`)

PROBLEM 22:

If the parameter `list` contains N elements, which best describes how many times the loop 7-10 and the loop 25-29 iterate. Note, this is not the runtime of each loop, but the number of iterations which is the same for each loop.

A. $O(\log N)$

B. $O(N)$

C. $O(N^2)$

D. none of the above

PROBLEM 23:

If the parameter `list` is `["a", "b", "c"]` which of the following best describes the variable `copy` after **one iteration** of the loop on lines 25-29 in method `duplicateB`?

A. `["a","b","c","a","b","c"]`

B. `["a","b","c","a","c","c"]`

C. `["a","b","c","b","c","c"]`

D. `["a","a","b","b","c","c"]`

PROBLEM 24:

The table below shows timings for calling both methods with `ArrayList` parameters with sizes ranging from 200,000 elements (returning a list with 400,000 elements) to 1,800,000 elements (returning a list with 3,600,000 elements).

size	duplicateA	duplicateB
200,000	0.48	0.005
400,000	1.86	0.003
600,000	4.18	0.023
800,000	7.42	0.037
1,000,000	11.60	0.046
1,200,000	16.88	0.068
1,400,000	22.96	0.082
1,600,000	30.38	0.096
1,800,000	38.15	0.095

Based on the code and these timings, which best characterizes the *runtime* of `duplicateA` when called with a parameter containing N values?

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$

PROBLEM 25:

Based on the code and these timings, which best characterizes the *runtime* of `duplicateB` when called with a parameter containing N values?

A. $O(1)$

B. $O(\log N)$

C. $O(N)$

D. $O(N^2)$