# Midterm 1: CompSci 201
# Form C

Prof. Astrachan

October 2, 2024

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**

### Farthest Shore

The code shown below is a correct implementation of the class `Person201Farthest` you were asked to write.

```
1  public class Person201Farthest {
       Run | Debug
2      public static void main(String[] args) throws Exception {
3          String file = "data/foodlarge.txt";
4
5          Person201[] people = Person201Utilities.readFile(file);
6          double max = 0;
7          Person201 a = null;
8          Person201 b = null;
9          for(int i = 0; i < people.length; i++){
10             for(int j = i+1; j < people.length; j++){
11                 double dist = Person201Utilities.distance(people[i],people[j]);
12                 if(dist > max){
13                     a = people[i];
14                     b = people[j];
15                     max = dist;
16                 }
17             }
18         }
19         System.out.printf("farthest distance is %3.2f between %s and %s\n",max,a.name(),b.name());
20     }
21 }
```

**PROBLEM 1:**

If $N$ is the number of elements in the array `people` assigned a value on line 5, what is the runtime of the code after line 5 (and thus not including line 5)? (note that the `distance` method does not depend on $N$.)

**A.** $O(N)$

**B.** $O(N^2)$

**C.** It depends on the values of latitude and longitude for the $N$ objects

**PROBLEM 2:**

If the loop on line 10 starts with `j=0` rather than `j=i+1` then which **one of** the following is true?

**A.** The big-Oh runtime will change.

**B.** The code will print the same thing.

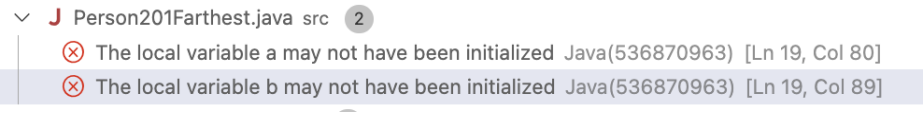
**C.** The code will print incorrect results since `i` will equal `j` in the loop body.

**PROBLEM 3:**

If the initialization of local variables `a` and `b` on lines 7 and 8 above is changed to what is shown below:

```
7       Person201 a;
8       Person201 b;
```

The program will not compile and generates the error shown




Note that the line number indicated in the error message for when the possibly uninitialized variables will cause an issue is the print statement on line 19. What is one situation when this would result in an error on line 19.

**A.** If the size of the array `people` is two and the objects in the array are in different hemispheres.

**B.** If the size of the array `people` is zero/0 an error will occur on line 19.

**C.** The actual error will occur on line 13, but the Java compiler generates error messages pessimistically.

**PROBLEM 4:**

The order of the statements on lines 13-15 can be changed as long as the statements remain in the `if(dist > max)` block and the code will still work correctly.

**A.** False, the order shown is the only correct order for correctness.

**B.** True, the three statements can be executed in any order and the code will be correct.

**PROBLEM 5:**

The code shown below is from a correct implementation of the `countEateries` method you were asked to write for the P0 Project; the body of the `for` loop is **not shown**.

```
11      /**
12       * Return the number of elements in people whose .eatery() value
13       * is equal to parameter eatery
14       * @param people is array of Person201 objects
15       * @param eatery is the searched for string
16       * @return the number of elements in people whose .eatery() value
17       * is equal to eatery
18       */
19      public int countEateries(Person201[] people, String eatery) {
20          int count = 0;
21          for(Person201 p : people) {…
26          return count;
27      }
```

Which statement below would make the code correct if the statement was the only statement in the loop body? Recall that all instance variables in `Person201` are private.

**A.** `if (p.eatery().equals(eatery)) count += 1;`

**B.** `if (p.eatery == eatery) count += 1;`

**C.** `if (eatery.equals(p.eatery)) count += 1;`

**D.** `if (p.eatery() == eatery) count += 1;`

**PROBLEM 6:**

Which of the following could replace the `for(Person201 p : people)` loop if the statement `Person201 p = people[k]` is added as the first statement of the loop body?

**A.** `for(int k=0; k < people.length; k++)`

**B.** `for(int k=0; k <= people.length; k++)`

**C.** `for(int k=0; k < people.length(); k++)`

**D.** `for(int k=0; k < people.size(); k++)`

Suppose that in Project P0, a new `Person201` method is added as shown below. This is from a modified *Person201.java* file since the method `distanceFrom` is new.

```
public double distanceFrom(Person201 other) {
    return Person201Utilities.distance(this,other);
}
```

Consider the four lines of code below code using `Person201` objects from Project P0 with a call of this new method.

```
Person201 p = new Person201("Sam", 38.6, 90.19, "Alpaca");
Person201 q = new Person201("Fred", 41.88, 87.63, "Panera");
double d = p.distanceFrom(q);
System.out.printf("%s to %s = %1.3f\n",p,q,d);
```

**PROBLEM 7:**

**For the call shown**: `p.distanceFrom(q)`, the code executed is

```
Person201Utilities.distance(this,other)
```

Which of the following is the value of `this` ?

**A.** `Person201 p`

**B.** `Person201 q`

**C.** It cannot be determined from the information given.

**PROBLEM 8:**

Suppose on the third line of the code shown above the assignment to `double d` is replaced by this code:

```
double d = q.distanceFrom(p);
```

Which *one of* the following is true? Note: that for any two `Person201` objects `p` and `q` that:

```
Person201Utilities.distance(p,q) == Person201Utilities.distance(q,p);
```

**A.** The output will be the same as with the original code.

**B.** The code will fail to compile and so will not run when changed as shown.

**C.** The value for distance will be the negative of what's printed before the change.

### NBody

The code below is from an implementation of the class `NBody.java` that passes all tests and works correctly to read data from a file to create and return an array of `CelestialBody` objects.

```
45  public static CelestialBody[] readBodies(String fname) throws FileNotFoundException {
46
47      Scanner s = new Scanner(new File(fname));
48
49          int np = s.nextInt();
50          CelestialBody[] bodies = new CelestialBody[np];
51          double radius = s.nextDouble();
52
53          for(int k=0; k < np; k++) {
54              double x = s.nextDouble();
55              double y = s.nextDouble();
56              double xv = s.nextDouble();
57              double yv = s.nextDouble();
58              double mass = s.nextDouble();
59              String fileName = s.next();
60              CelestialBody p = new CelestialBody(x,y,xv,yv,mass,fileName);
61              bodies[k] = p;
62          }
63          s.close();
64          return bodies;
65
66  }
```

The body of the `for` loop can be replaced by a single statement without *generally affecting* the correctness of the implementation.

```
bodies[k] = new CelestialBody(s.nextDouble(),s.nextDouble(),s.nextDouble(),
                              s.nextDouble(),s.nextDouble(),s.next());
```

**PROBLEM 9:**

Which one of the following is *true*?

**A.** The call of `s.nextDouble()` on line 51 will cause an error when a file has more than 100 lines of data when combined with the new version, but not with the original version.

**B.** The fifth parameter to the constructor, `mass` in the original version, cannot be read by `s.nextDouble()` since it is possible that the value is 25, e.g., does not have a decimal point in the file.

**C.** The new code, with calls of `s.nextDouble()` and `s.next()` in the call of the constructor works in all cases that the original code works.

**PROBLEM 10:**

In the code shown, line 51 could be replaced by the call `s.nextDouble()`, with no assignment to a variable, and the code's behavior will not change.

**A.** True, no assignment to a variable is necessary for the code to work

**B.** False, assigning the value to a variable is necessary for the code to work

**PROBLEM 11:**

The class `CelestialBody` has methods `getX()` and `getY()` – these methods are shown in the code below from a student implementation of `calcDistance`, a method in the `CelestialBody` class.

```
94 public double calcDistance(CelestialBody b) {
95         double rootThis = Math.pow((myXPos - b.getX()), 2) + Math.pow((myYPos - b.getY()), 2);
96         return Math.sqrt(rootThis);
97 }
```

Which one of the following is *false*

**A.** The expression `b.getX()` can be replaced by `b.myXPos` without affecting the program.

**B.** The expression `myXPos` can be replaced by `this.myXPos` without affecting the program.

**C.** The expression `myXPos` can be replaced by `getX()` without affecting the program.

**D.** This code will not work unless `import java.lang.Math` is included at the beginning of the file.

**PROBLEM 12:**

In the class `CelestialBody` there is one six parameter constructor as shown:

```
31     public CelestialBody(double xp, double yp, double xv,
32                          double yv, double mass, String filename){
```

You're asked to make a two-parameter constructor with a mass (double) and a file-name (string) that sets all the other variables to zero. For example, the code below with this new constructor would create a body with a heavy mass at the center (0,0) of the universe with zero x- and y-velocity.

```
CelestialBody bod = new CelestialBody(1e40,"blackhole.png")
```

Which of the following can be the body of the new constructor?

**A.** `this(0,0,0,0,mass,filename)`

**B.** `this(mass,filename)`

**C.** `this(0,0,mass,filename)`

**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 13:**

If two string variables `s` and `t` have the same 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 14:**

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

**A.** True

**B.** False

**C.** Cannot be determined from the information given

**PROBLEM 15:**

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

### Inner Visions

The class `Inner` encapsulates a `String`, but instruments the overridden methods `hashCode` and `equals` with print statements to help understand when these methods are called.

You'll answer four questions in which this class is used.

```
6  public class Inner {
7      private String myString;
8      public Inner(String s){
9          myString = s;
10     }
11     @Override
12     public int hashCode(){
13         System.out.printf("%s hashcode called\n",this);
14         return myString.hashCode();
15     }
16     @Override
17     public boolean equals(Object o){
18         Inner thing = (Inner) o;
19         System.out.printf("this %s equals %s?\n",this,thing);
20         return myString.equals(thing.myString);
21     }
22     @Override
23     public String toString(){
24         return myString;
25     }
26 }
```

**PROBLEM 16:**

Which one of the following is **false** about the use of `@Override` before the three methods it appears?

**A.** `@Override` indicates that the inherited method from class `Object` will be *overridden* or implemented in the class `Inner`.

**B.** `@Override` can help the compiler check against inadvertent coding problems: e.g., if `tostring` was used instead of `toString` or if the parameter to `.equals` had type `Inner` instead of `Object`.

**C.** `@Override` starts with the letter capital 'O' because it's a big-Oh indication.

**PROBLEM 17:**

What does this code print?

```
Inner thing = new Inner("Duke");
System.out.println(thing);
```

**A.** An error message because the code either does not compile or does not run

**B.** Inner@Duke

**C.** Duke

**D.** @17a7cec2 (or similar, a memory address for the object `thing`)

**PROBLEM 18:**

Consider the method `runHashing` shown below and this call:

```
runHashing(new String[]{"hashing","greendance","duke"}
```

```
38  public void runHashing(String[] words){
39      HashMap<Inner,Integer> map = new HashMap<>();
40      for(String s : words) {
41          Inner elt = new Inner(s);
42          if (map.putIfAbsent(elt, elt.hashCode()) != null) {
43              System.out.printf("already contains %s\n",elt);
44          }
45      }
46      System.out.printf("size of keys = %s\n",map.keySet().size());
47      HashSet<Integer> set = new HashSet<>(map.values());
48      System.out.printf("size of value set = %s\n",set.size());
49  }
```

This is the ouput generated by the call.

```
hashing hashcode called
hashing hashcode called
green dance hashcode called
green dance hashcode called
duke hashcode called
duke hashcode called
size of keys = 3
size of value set = 3
```

Which *one* of the following is **not correct** about what's printed by the method `Inner.hashCode` as shown in the output?

**A.** There is an explicit call of `elt.hashCode` on line 42 that generates a line of the output shown.

**B.** The print statement in class `Inner` on line 13 will call the `Inner.toString` method.

**C.** The call of `map.putIfAbsent` on line 42 generates a call (in the `HashMap.putIfAbsent` code) of `Inner.hashCode` to determine if `elt` is already in the map and to add it to the map if not already there.

**D.** One call of `Inner.hashCode` prints two lines of output as seen in the output above.

**PROBLEM 19:**

If lines 41-44 in `runHashing` are replaced by the lines below in which `containsKey` is used, what is true about the output generated by the same call of the method `runHashing`?

```
runHashing(new String[]{"hashing","greendance","duke"}
```

```
41          Inner elt = new Inner(s);
42          if (! map.containsKey(elt)){
43              map.put(elt,elt.hashCode());
44          }
45          else {
46              System.out.printf("already contains %s\n",elt);
47          }
```

**A.** The output will be the same as when the `putIfAbsent` code is used.

**B.** There will be more lines of output generated because `Inner.hashCode` will be called more often in this new code.

**PROBLEM 20:**

If the version of `runHashing` with `putIfAbsent` is called with a different array as shown:

```
runHashing(new String[]{"unheavenly","hypoplankton"});
```

The output changes to the following:

```
unheavenly hashcode called
unheavenly hashcode called
hypoplankton hashcode called
hypoplankton hashcode called
this hypoplankton equals unheavenly?
size of keys = 2
size of value set = 1
```

What best explains this output?

**A.** The value of `"unheavenly".hashCode()` is the same as the value of `"hypoplankton".hashCode()`

**B.** The value of `"unheavenly".hashCode()` is different than the value of `"hypoplankton".hashCode()`

**C.** Hypoplankton is a tiny organism found just above the ocean floor.

**PROBLEM 21:**

Which of the following best characterizes the runtime complexity of the call `calcSmall(arr)` where the array `arr` contains $N$ elements?

```
 8   public String calcSmall(String[] arr){
 9       String small = "";
10       for(String s : arr) {
11           if (s.length() < small.length()) {
12               small = s;
13           }
14       }
15       return small;
16   }
```

**A.** $O(N)$

**B.** $O(N \log N)$

**C.** $O(N^2)$

**PROBLEM 22:**

Suppose that in the code above for method `calcSmall` that the `for` loop on lines 10-14 is copied and then pasted as lines 15-19 and again as lines 20-24 so that the same three loops comprise lines 10-24, with the return statement after the three loops. What is the runtime complexity of the changed code?

**A.** The same answer, the asymptotic complexity stays the same.

**B.** A different answer, the asymptotic complexity changes.

**PROBLEM 23:**

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

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

**A.** $O(\log N)$

**B.** $O(N)$

**C.** $O(N \log N)$

**D.** $O(N^2)$

**PROBLEM 24:**

What is the runtime complexity of the call `triple(N)` for the code below?

```
20    public int triple(int n){
21        int sum = 0;
22        for(int j=0; j < n/100; j += 1){
23            for(int k=0; k < n/100; k += 1){
24                for(int h=0; h < n/100; h += 1){
25                    sum += j+k+h;
26                }
27            }
28        }
29        return sum;
30    }
```

**A.** $O(N)$

**B.** $O(N^2)$

**C.** $O(N^3)$

**PROBLEM 25:**

Consider the output of the code below

```
System.out.println(triple(1000));
System.out.println(triple(1000) < triple(100000));
```

for the method `triple` shown above. The output printed is

```
13500
false
```

Which one of the following is *false*?

**A.** The call `triple(100)` returns 0

**B.** The call `triple(100000)` returns a value less or equal to 13500

**C.** The value of `Integer.MAX_VALUE + 1` is a positive number.

**PROBLEM 26:**

The code below results in printing `[1, 2, 4, 5, 10, 20, 25, 50, 100]` as a result of printing the value of `divisors(100)`:

```
List<Integer> list = divisors(100);
Collections.sort(list);
System.out.println(list);
```

The method `divisors` is shown below:

```
41  public List<Integer> divisors(int n){
42      ArrayList<Integer> list = new ArrayList<>();
43      for(int k=1; k*k <= n; k++){
44          if (n % k == 0){
45              list.add(k);
46              if (k*k != n) list.add(n/k);
47          }
48      }
49      return list;
50  }
```

What is the asymptotic runtime of the call `divisors(N)`?

**A.** $O(\sqrt{N})$

**B.** $O(N)$

**C.** $O(N^2)$

**D.** $O(N^3)$

**PROBLEM 27:**

Which one of the following is *false*?

**A.** The size of the array returned by the call `divisors(N)` has at most the same big-Oh complexity as the runtime of the call.

**B.** There is a value $N$ such that `divisors(N).size() == 2`

**C.** Removing line 46 from the method changes the asymptotic runtime of the call `divisors(N)` because the size of the returned array is reduced by half.

The next questions deal with a `Map<String,String> acc` that initializes a map to contain each university of the ACC (Atlantic Coast Conference) as a key and the state in which the university is located as the value:

```
18      Map<String,String> acc =  new HashMap<>();
19      acc.put("Duke", "North Carolina");
20      acc.put("Wake Forest", "North Carolina");
21      acc.put("UNC", "North Carolina");
22      acc.put("NC State", "North Carolina");
23      acc.put("UVA", "Virginia");
24      acc.put("Virginia Tech", "Virginia");
25      acc.put("Florida State", "Florida");
26      acc.put("University of Miama", "Florida");
27      acc.put("Stanford", "California");
28      acc.put("Cal/Berkeley", "California");
29      acc.put("Syracuse", "New York");
30      acc.put("Pittsburgh", "Pennsylvania");
31      acc.put("Boston College", "Massachusetts");
32      acc.put("Clemson", "South Carolina");
33      acc.put("Louisville", "Kentucky");
34      acc.put("SMU", "Texas");
35      acc.put("Georgia Tech", "Georgia");
36      acc.put("Notre Dame", "Indiana");
```

Given this initialization, the code below prints the numbers 18 and 12, in that order.

```
System.out.println(acc.values().size());
System.out.println(new HashSet<String>(acc.values()).size());
```

**PROBLEM 28:**

Which best explains why the numbers printed are different, i.e., 18 and 12?

**A.** Some states contain more than one university from the ACC.

**B.** Some universities have the same `hashCode`

**C.** Some of the state names are misspelled.

A method `revMap` is intended to return a map in which keys are names of states and the corresponding value is an ArrayList of the universities in that state, e.g., some of the map entries in this returned map are:

```
North Carolina   [NC State, Duke, UNC, Wake Forest]
New York         [Syracuse]
California       [Cal/Berkeley, Stanford]
```

```
18  public static Map<String,ArrayList<String>> revMap(Map<String,String> acc){
19      Map<String,ArrayList<String>> ret = new HashMap<>();
20      for(String uni : acc.keySet()) {
21          String state = acc.get(uni);
22          if (! ret.containsKey(state)) {
23              // statement 1
24          }
25          // statement 2
26      }
27      return ret;
28  }
```

**PROBLEM 29:**

Bubble in the choice A and fill in the code for statement 1 in the space on the back of the answer sheet so that the code works correctly.

**PROBLEM 30:**

Bubble in the choice A and fill in the code for statement 2 in the space on the back of the answer sheet so that the code works correctly.

Consider a method `filter` such that the code below prints `["green","orange","magenta"]` since `filter` returns an array of the strings in its first parameter whose length is greater than or equal to its second parameter. The strings `"red"` and `"blue"` are not in the returned array because the length of each is less than 5. The other strings each have length greater than or equal to 5.

```
String[] colors = {"green","red","blue","orange","magenta"};
String[] big = filter(colors,5);
System.out.println(Arrays.toString(big));
// prints ["green","orange","magenta"]
```

**PROBLEM 31:**

What is the smallest value that can replace 5 in the call of `filter` shown above so that the returned array is empty?

**A.** 6

**B.** 7

**C.** 8

**D.** 9

Here is code for `filter` that would be correct if two lines are added.

```
10  public String[] filter(String[] words, int minLength){
11      // statement 1
12      for(String s : words){
13          if (/** expression 1 */){
14              list.add(s);
15          }
16      }
17      return list.toArray(new String[0]);
18  }
```

**PROBLEM 32:**

Bubble in the choice A and fill in the code for *statement 1* on line 11 in the space on the back of the answer sheet so that the code works correctly.

**PROBLEM 33:**

Bubble in the choice A and fill in the code for *expression 1* on line 13 in the space on the back of the answer sheet so that the code works correctly.

The method `maxLetter` shown below is intended to determine what character in a String occurs more frequently than any other (assume this character is unique, there is one maximally occurring character). For example, a call and the output generated are shown:

```
maxLetter("the quick brown fox jumps over the lazy dog");
char 'o' occurs 4 times
```

```
30  public static void maxLetter(String s) {
31      int[] counts = new int[256];
32      for(char ch : s.toLowerCase().toCharArray()) {
33          // statement 1
34      }
35      int max = 0;
36      for(char ch = 'a'; ch <= 'z'; ch++){
37          if (/* expression */) {
38              max = counts[ch];
39          }
40      }
41      for(char ch = 'a'; ch <= 'z'; ch++){
42          if (counts[ch] == max) {
43              System.out.printf("char '%s' occurs %d times\n",ch,max);
44          }
45      }
46  }
```

In the method, the array `counts` acts as a map such that `counts[ch]` is the number of times `ch` occurs for any char value `ch`.

**PROBLEM 34:**

Bubble in the choice A and fill in the code for statement 1 on line 33 in the space on the back of the answer sheet so that the code works correctly. The statement *must* reference the array `counts`.

**PROBLEM 35:**

Bubble in the choice A and fill in the code for *expression* on line 37 in the space on the back of the answer sheet so that the code works correctly. The statement *must* reference the array `counts`.