COMP110 Final Study Guide

Key terms

1. OOP

--- definition:

fields, methods, constructors

this

--- usage

main method (constructing objects, method calls, change of fields)

parameters vs arguments

local variables vs fields

expressions (method chaining & nested methods)

null

2. Other Basics

datatypes (primitive, String, objects)

reference and primitive types

operators (arithmetic, boolean, concatenation)

conditionals

2D arrays & nested loops (new!)

arrays

loops (for & while)

interfaces

recursion

ADT (List, Map)

simple insertion sort and binary search

1. T/F:

a. A condition is required for an else statement.

b. A condition is required for an else-if statement.

c. Parameters is to method calls as arguments is to method declarations.

d. == is to comparison as = is to assignment.

e. Not every element in the 2D array has to be the same variable type.

f. Array indices must be of type `int` and can be a variable or expression.

g. The correct way to get the width and height of a 2D array called arr is arr.length and arr.height.

h. The correct way to get the width and height of a 2D array called arr is arr.length and arr[0].length.

i. A constructor of a class always has the same name as the class name.

j. A constructor has no return type.

2. Does each the following values evaluate to true or false?

```
_____true && true          _____true && false     _____false && false
_____true || false         _____false || false
_____("a"+1+2).equals("a3")  _____3/4 == .75 || 3%4 == 1
```

3. Read the code below, then answer the question.

```
for (int i = 4; i >= 1; i--) {

    if (i == 10) {
        System.out.println("A");
    } else if (i == 2) {
        System.out.println("B");
    } else if (i >= 4) {
        System.out.println("C");
    } else {
        System.out.println("D");
    }

    System.out.println(i);
}
```

What is the output of the code on the right? Write out one line per box (use as many boxes as needed).

4. Correct the error(s) in each snippet of code (*mistakes in logic and implementation are not errors in these exercises*). To the right, explain the error(s) and name the concept(s) emphasized for each snippet of code.

a

```
public int sum(int[] arr) {

 for (int a = 0; a < arr.length; a++) {
   int sum = 0;
   sum = sum + arr[a];
 }

 return sum;
}
```

b

```
public boolean compare(String[] arr, List<String> list) {
 if (arr.length != list.length) {
    return false;
 } else {
    int i = 0;
    for (String s : list) {
       if (s != arr[i]) {
           return false;
       }
       i++;
    }
 }

 return true;
}
```

c

```
Map<String, Double> map = new HashMap<String, Double>();
map.add(“Helen”, 10.0);
map.add(“Dorian”, 6.77);
map.remove(6.77);
```

5. A 2D array is a 1D array whose elements are also 1D arrays. Bear with that and try to explore this further with the following 3 questions.

a. Declaration and Instantiation

Sandy is a diligent squirrel who has collected a number of acorns and seeds for the approaching winter. Much to her delight, Sandy just found a rectangular lawn where she can bury a certain amount of food in each slot of the lawn. Go ahead and construct a 2D array for this rectangular lawn as Sandy's food storage map in the constructor of the following class *Squirrel*.

```
public class Squirrel{
 private double[][] lawn; //The 2D array of type double has been declared

 public Squirrel(int row, int col, double food){
/*The number of rows and columns of the lawn and the total amount of grains
Sandy has collected as winter food were given as parameters
*/

/*(1): Instantiate the array lawn as Sandy's food storage map*/




/*(2): Use a loop to fill in the elements of the lawn where each slot
contains the same amount of food
*/










}
}
```

b. Using 2D array with a while loop

Last night, a powerful storm slammed Chapel Hill and some slots of Sandy's food storage lawn were damaged unfortunately. So, the amount of food in each remaining slot decreased due to the flood. As a warmhearted COMP 110 student, you want to help Sandy calculate how much food in total was left under the lawn so that Sandy can better prepare for the next round of grain collecting. Here is an illustration for the before and after storm scenario.

Before Storm: 3*4 Rectangle Lawn	After Storm: Left Lawn
3.2 | 3.2 | 3.2 | 3.2	1.2 | 0.0 | 0.0 | 0.0
3.2 | 3.2 | 3.2 | 3.2	1.0 | 2.2 | 3.2 | 1.1
3.2 | 3.2 | 3.2 | 3.2	2.7 | 3.2 | 0.0 | 2.1
Total amount of food=38.4	Total amount of food=16.7

Now, a 2D array named *leftLawn* of the element type double is given as the parameter of the method *afterStorm* that returns a double indicating the total amount of food left under the lawn. Each element of *leftLawn* represents the amount of food left at that position. Go ahead and finish the *afterStorm* method.

```
public class Squirrel{
 private double[][] lawn;

 Public Squirrel(){
   /****as in question a ****/
 }

 public double afterStorm(double[][] leftLawn){
       double totalFood=0;















 }
}
```

6. This is broad practice in object-oriented programming (OOP), and it uses several concepts you've learned throughout the semester. You will implement classes and an interface that represent customers and a brand new pizza restaurant on Franklin Street, then use them in a main method - all from scratch. You can imagine how a program like this might be useful for a company's operations management. Try to do this without the help of Eclipse!

a. Write a class called **Customer** that declares the following fields, constructor, and methods:

A. Fields (name; visibility; type; *description*):
   1. _funds; private; double; *dollars to spend*
   2. _hunger; private; double; *hunger should range from 0.0 to 1.0*

B. Constructor (parameters; *description*):
   1. double funds; double hunger; *initializes fields and corrects _hunger to be within the range 0.0 to 1.0 if the parameter* hunger *is out of range*

C. Methods (name; visibility; return type; parameters; *description*):
   1. eat; public; boolean; double amount; *if _hunger is greater than 0.0, lowers _hunger by the amount of food in the parameter and returns true, returns false otherwise to signify customer is "full"*
   2. spend; public; boolean; double amount; *lowers _funds by the amount spent in the parameter ONLY if doing so will not result in negative _funds, returns true if money is spent, returns false otherwise*

b. Write an interface called **Restaurant** that declares the following public methods:

D. Methods (name; return type; parameters):
   1. serveGuests; void; none

c. Write a **Buffet** class that implements the Restaurant interface above:

E. Fields (name; type; visibility; *description*):
   1. _guestList; List<Customer>; private; list of guests who've paid to eat

F. Constructor (parameters; *description*):
   1. none; *initialize* _guestList *to a new empty* ArrayList *that holds* Customers

G. Methods (name; visibility; return type; parameters; description)
   1. addGuest; public; void; Customer guest; *have the guest spend* 16.0 *then add* guest *to* _guestList *only if* 16.0 *was successfully spent*
   2. removeGuest; public; void; Customer guest; *remove* guest *from* _guestList - *hint: to do this, use the* indexOf *and* remove *methods from the* List *interface*
   3. serveGuests; (see details in Restaurant interface); *while there are guests present, use a for-each loop to go through the guests and have each eat* 0.3 *in food, and if any appear full, remove that guest from the guest list*

d. Write a class called **Runner** that includes a main method. In the main method, perform the following actions:

H. Declare and instantiate a Restaurant using the Buffet class. Call it "pizzaCorner."

I. Declare and instantiate three Customer objects with appropriate *funds* and *hunger* levels of your choosing - here are suggested variable names:
   1. "undergrad"
   2. "grad"
   3. "prof"

J. Add the Customer objects to pizzaCorner's guestList. Will all be added successfully?

K. Have pizzaCorner serve the guests.

7. Given the class below, answer the questions.

```
public class ClassA {
   private int _a;
   private double _b;
   private String _c;

   public ClassA(int a, double b, String c) {
      _a = a;
      _b = b;
      _c = c;
   }

   public double getA() {
      return a;
   }

   public void foo(String s) {
      _c = _c + s;
   }

   public int addAll(int[][] nums) {
      // implement the method below







   }
}
```

(a) Circle the constructor of the class.

(b) Method `getA` (in bold) is implemented incorrectly. Fix the error(s).

(c) Implement the `addAll` method, such that the method returns the sum of all values in `nums`.

(d) Circle the correct implementation for each line of code in the main method below.

```
public static void main(String[] args) {
```

`ClassA a = ClassA(1, 2.0, “.”);`	`ClassA a = new ClassA(1, 2.0, “.”);`
`String str = “c”;`	`String str = ‘c’;`
`a.foo();`	`a.foo(str);`
`getA();`	`a.getA();`
`int[][] ints = new int[][];`	`int[][] ints = new int[3][5];`
`ints[2][2] = 8;`	`ints.set(2,2,8);`
`System.out.println(ints[1][1]);`	`System.out.println(ints.get(1,1));`
`System.out.println(ints.length);`	`System.out.println(ints.size());`
`int[][] out = a.addAll(ints);`	`int out = a.addAll(ints);`
`System.out.println(out);`	`System.out.println(int out);`

```
}
```

Then explain why each of the other choice is wrong:

(e) For the main method code in (d), suppose we set every element in `ints` to be 8. What will be the printed output?

8. In this problem, we'll be modeling the Conway's Game of Life simulation. The universe of the Game of Life is a two-dimensional grid containing cells, which can be either alive or dead. Each cell interacts with its eight neighbors that are adjacent to it horizontally, vertically and diagonally.

a. Here is the definition of the Cell method. Complete the *copy* method such that it returns a new Cell with the same state.

```
public class Cell {
 boolean _isAlive;
 public Cell(boolean isAlive) {
   _isAlive = isAlive;
 }

 public boolean isAlive() {
   return _isAlive;
 }

 public void dead() {
   _isAlive = false;
 }

 public void alive() {
   _isAlive = true;
 }

 public Cell copy() {
   // 1






 }
}
```

The following class contains the grid containing the cells.

```
public class GameOfLife {
 Cell[][] _cells;
 public GameOfLife(Cell[][] cells) {
   _cells = cells;
 }

 public int getWidth() { … }
 public int getHeight() { … }
 public Cell getCell(int x, int y) {...}
 public Cell[][] copy() { … }
 public int getLiveNeighbourCount(int x, int y) { … }
 public boolean shouldLive(int x, int y) { … }
 public void nextState() {...}

}
```

b. Complete the *getWidth* method so that it returns the width of the grid.

```
 public int getWidth() {


 }
```

c. Complete the *getHeight* method so that it returns the height of the grid.

```
 public int getHeight() {

 }
```

d. Complete the *getCell* method so that it returns the Cell at the position (x, y).

```
 public Cell getCell(int x, int y) {

 }
```

e. Complete the *copy* method, which copies the current grid of cells to a new grid and returns that copy.

```
public Cell[][] copy() {
  // 4







}
```

f. Complete the *getLiveNeighbourCount* method, which returns the number of neighboring cells that are alive. Currently the for loop iterates through all 8 neighboring cells as well as the cell itself. You will need to check whether each (i, j) coordinates is valid (i.e. not out of index and not equal to (x,y)) and update the *count* if you see a cell that is alive.

```
public int getLiveNeighbourCount(int x, int y) {
  int count = 0;
  for (int i = x - 1; i <= x + 1; i++) {
    for (int j = y - 1; j <= y + 1; j++)  {






    }
  }

  return count;
}
```

7. Complete the *shouldLive* method. If a cell is currently alive, then it should live if it has 2 or 3 alive neighbors. If the cell is currently dead, it should live if it has exactly 3 alive neighbors.

```
public boolean shouldLive(int x, int y) {




}
```

8. Complete the *nextState* method, which simulates a step in time. Iterate through all of the cells and update its state using the *alive* and *dead* methods defined in *Cell*. In the actual game, all of the cells transition to their next state spontaneously. Be sure to create a copy of the grid and modify that copy, since we are simulating all of the updates happening at once.

```
 public void nextState() {









 }

}
```

9. Calculate the value of the variable or expression indicated.

a. What is the value of the variable `d` after the following code executes?

```
List<Integer> a = new ArrayList<Integer>();
a.add(1);
a.add(2);
a.add(3);
List<Integer> b = new ArrayList<Integer>();
b.add(4);
b.add(5);
b.add(6);
List<Integer> c = new ArrayList<Integer>();
c.add(a.get(0));
c.add(b.get(1));
c.add(a.get(2));
b = a;
a = c;
c = b;
int d = a.get(0) + b.get(1) + c.get(2);
```

b. What is the value of `bar` after the code below executes?

```
int foo = 30;
int bar = 15;
if (foo < 90) {
 bar += 10;
 foo *= 3;
} else if (foo < 80) {
 bar += 5;
} else if (foo > 70) {
 bar -= 5;
}
```

c. Given the following method definition of foo:

```
public int foo(int a, int b) {
      if (b < a) {
        return foo(b, a);
      }
      if ((b / a) * a == b) {
        return a;
      }
      return foo (a, b - a);
}
```

What is the value of calling `foo(2,7)` on an object of the class that this method belongs to?