Name _____

CISC 3115 – Modern Programming Techniques
Fall 2024
Exam 1 Solutions

Part I. Answer all of the following (70 points)

1. (**10 points**) Write a `Counter` class with the following:

   - a single integer instance variable initialized to `0`
   - `up` and `down` methods
   - a `toString` method

   please make sure to use `public` and `private` properly on the members of the class.

```
class Counter {
  public void up() {val++;}
  public void down() {val--;}
  public String toString() {return val;}

  private int val = 0;
}
```

2.

a. (**4 points**) A `Container` class has a `find` method that accepts a value as an argument and returns the location of that value in the container, or `-1` if it's not there. Write the method `contains` (also a member of the class) that also accepts a value as an argument but returns a `boolean` indicating whether the value is in the container or not. You know nothing else about the container's internals so `contains` should be written by leveraging `find`.

```
public boolean contains(int val) {return find(val) != -1;}
```

b. (**6 points**) Write some code that creates a `Container` object, adds the numbers `1` through `10` to the container (using a `for` loop), prints the container (using `toString`), replaces the first element of the container with the number `100`, and finally prompts the user at the keyboard for an integer (assume a `Scanner` object has already been declared and created), and prints out whether the number is in the container. The `Container` class has the usual `add`, `toString`, `find`, `contains`, `get`, and `set` methods.

```
Container c = new Container();

for (int i = 1; i <= 10; i++)
      c.add(i);
System.out.println(c);
c.set(0, 100);
System.out.print("Enter a number to search for: ");
int num = scanner.nextInt();
if (c.contains(num)
      System.out.print(num + " is in the container");
else
      System.out.print(num + " is not in the container");
```

3.

a. (**4 points**) Write two constructors for a `Color` class with `r`, `g`, `b` integer instance variables. The first constructor should accept three integer arguments and use them to initialize the instance variables. The second is a default constructor that initializes the `Color` to black (0 intensity for all three color components) by leveraging the 3-argument constructor.

```
Color(int r, int g, int b) {
      This.r = r;
 This.g = g;
 This.b = b;
}

Color() {this(0, 0, 0);}
```

b. (**3 points**) Write the method `isGrey` for the `Color` class (a color is grey if all three color components have the same intensity).

```
public boolean isGrey() {return r = g && g == b;}
```

c. (**3 points**) Declare a class variable (i.e., only one copy for the `Color` class) named `GREEN`, that references a `Color` object (which you create as well) representing pure green.

```
public static Color GREEN = new Color(0, 255, 0);
```

final is ok here… or not

4.

a. (**4 points**) Write the `toString` method for a `Quadrilateral` class implemented using four `Point` instance variables `p1, p2, p3,` and `p4`. The methods should print out the four variables separated by commas.

```
public String toString() {
      return p1 + ", " + p2 + ", " + p3 + ", " + p4;
}
```

b. (**6 points**) The `Line` class accepts two `Point` objects for its constructor, and has a `length` method. Write the method `isRhombus` of the above `Quadrilateral` class, that creates four `Line` objects (`side1-side4`) corresponding to the sides of the figure, and returns true if all four sides are equal in length. (Assume the sides are `p1-p2, p2-p3, p3-p4,` and `p4-p1`)

```
public boolean isRhombus() {
 Line
      Side1 = new Line(01, o2),
      Side2 = new Line(p2, p3),
      Side3 = new Line(p3, p4),
      Side4 = new Line(p4, p1);

 return
      side1.lenght() == side2.length() &&
      side2.length() == side3.length() &&
      side3.length() == side4.length() &&
      side4.length() == side1.length();
}
```

5.

a. (**5 points**) Assume the existence of a `PhonebookEntry` class containing the methods `getName` and `getNumber` (no first name). Declare the instance variables of the `Phonebook` class (implemented as a partially populated array of `PhonebookEntry`, i.e., an array and a size)

```
private int size = 0;
private PhonebookEntry [] entries = new PhonebookEntry[100];
```

b. (**5 points**) Write the `reverseLookup` method of the `Phonebook` class that accepts a phone number (`String`) and returns the corresponding name, or `null` if the number is not found (use the instance variables name of part *a*).

```
public String reverseLookup(String number) {
    for (int i = 0; i < size; i++)
        if entries[i].getNumber.equals(number)
           return entries.getName();
    return null;
}
```

6.

a. (**7 points**) Write an immutable integer class containing a single integer value, a constructor that initializes that value to its argument, and the methods `add`, `mul`, and `toString` (`sub` or `div` methods do not need be written).

```
class ImmInt {
      ImmInt(int val) {this.val = val;}

      ImmInt add(int other) {return new ImmInt(val + other.val;}
      ImmInt mul(int other) {return new ImmInt(val * other.val;}

      public String toString() {return val + " ";}
}
```

b. (**3 points**) Declare three objects of this class, initialized to `1`, `2`, and `3`, and print the result of multiplying the sum of the first two by the third

```
ImmInt
     imm1 = new ImmInt(1),
     imm2 = new ImmInt(2),
     imm3 = new ImmInt(3);
System.out.println(imm1.add(imm2).mul(imm3)); // … or imm3.mul(imm1.add(imm2));
```

7.

a. (**5 points**) Write (just) the method `increaseBy` of a mutable integer class (again containing a single integer value).

```
public MutInt increaseBy(MutInt other) {
      val += other.val;
      Return this;
}
```

b. (**5 points**) Declare three objects of this class, initialized to 1, 2, and 3, and print the result of multiplying the sum of the first two by the third. What are the values of the three objects after this operation.

```
MutInt
    mut1 = new MutInt(1),
    mut2 = new MutInt(2),
    mut3 = new MutInt(3);
System.out.println(imm1.add(imm2).mul(imm3)); // or imm3.mul(imm1.add(imm2));
```

Part II. Answer one of the following two questions (20 points):

8. Given the following app (the `main` is in the `CarDemo` class):

```
class Engine {
     public Engine(boolean isHybrid, int numCylinders) {
          this.isHybrid = isHybrid;
          this.numCylinders = numCylinders;
     }
     public Engine() {this(false, 4);}

     public int mpg() {return 60/numCylinders;}

     public String toString() {
          return numCylinders + " cylinder" + (isHybrid ? " hybrid" : "");
     }

     private boolean isHybrid;
     private int numCylinders;
}
```

```
class Car {
     public Car(int year, String model,  Engine engine) {
          this.year = year;
          this.model = model;
          this.engine = engine;
     }
     public Car(String model) {this(2018, model, new Engine());}

     public int mpg() {return engine.mpg();}

     public String toString() {
          return "a " + year + " " + model + " " + engine;
     }

     private int year;
     private String model;
     private Engine engine;
}
```

```
class CarDemo {
     public static void main(String [] args) {
          Engine engine = new Engine(true, 6);
          Car car1 = new Car(2017, "Impala", engine);
          System.out.println("car1: " + car1);
          System.out.println("car1's mpg: " + car1.mpg());

          Car car2 = new Car("Malibu");
          System.out.println("car2: " + car2);
          System.out.println("car2's mpg: " + car2.mpg());
     }
}
```

a. (**10 points**)What is output by the program's execution?

```
car1: a 2017 Impala 6 cylinder hybrid
car1's mpg: 10 mpg
car2: a 2018 Malibu 4 cylinder
car2's mpg: 15
```

b. (**5 points**) Which method is a *delegation method*? What makes it a delegation method?

The `mpg` method of the `Car` class … all it does is call the `mpg` method of `Engine`

c. (**5 points**) Can you see a problem with an `Engine` being created with zero (0) cylinders (it's more than simply – *'a Car can't have 0 cylinders'*)? What is the problem, and what would you do about it (nothing detailed—just a sentence is fine)?

`mpg` divides 60 by the number of cylinders… for 0 cylinders this would result in a 0-divide

9. (**20 points**) One variation of the `PhonebookEntry` class of Lab 2.5 is to represent the phone number as a class, `PhoneNumber` with three integer instance variables for the individual components: *area-code*, *exchange* and *line number* (for example, the number (718) 951-5000 consists of the area code 718, the exchange 951, and the line number 5000). Here is a specification for such a class

- State
  - three integer instance variables representing the area code, exchange, and line-number of the phone number
- Behavior
  - a constructor that accepts three integers representing the area code, exchange, and line number respectively
  - a constructor that accepts two integers – the exchange and line number, and sets the area code to 800. *To receive full credit for this, you must invoke the three parameter constructor.*
  - `getAreaCode`, `getExchange`, and `getLineNumber` methods
  - an `equals` method that returns whether two `PhoneNumber` objects are equal (i.e., their area code, exchange, and line numbers are all equal). The method header is

    ```
    boolean equals(PhoneNumber otherNumber)
    ```

    (notice the header is similar to that of `add` of the immutable integer class; the logic is similar, but you are comparing the instance variables of the receiver and parameter rather than adding them).
  - A `toString` method that returns the phone number in the format `(nnn)nnn-nnnn`.
  - a (`static`) `read` method that reads in three integers from the supplied `Scanner` (no prompts please), and uses them to create and initialize a `PhoneNumber` object (using the three-parm constructor), and returns the new object. The method header is

    ```
    static PhoneNumber read(Scanner scanner)
    ```

Provide the class definition of the `PhoneNumber` class. Make sure you include `public` and `private`, and use them appropriately.

```
import java.util.Scanner;

class PhoneNumber {
   PhoneNumber(int areaCode, int exchange, int lineNumber) {
       this.areaCode = areaCode;
       this.exchange = exchange;
       this.lineNumber = lineNumber;
   }
   PhoneNumber(int exchange, int lineNumber) {
       this(800, exchange, lineNumber);
   }

   public int getAreaCode() {return areaCode;}
   public int getExchange() {return exchange;}
   public int getLineNumber() {return lineNumber;}

   public boolean equals(PhoneNumber other) {
       return areaCode== other.areaCode &&
                 exchange == other.exchange &&
                 lineNumber == other.lineNumber;
       }

   public static PhoneNumber read(Scanner scanner) {
       if (!scanner.hasNextInt()) return null;
       int
             areaCode = scanner.nextInt(),
             exchange = scanner.nextInt(),
             lineNumber = scanner.nextInt();
       return new PhoneNumber(areaCode, exchange, lineNumber);
   }

   public String toString() {
       return "(" + areaCode + ")" + exchange + "-" + lineNumber;
   }

   private int areaCode, exchange, lineNumber;
}
```

Part III (10 points). Answer what you can and/or want to.

10. Did Lab 0 help you warm up in terms of getting back into coding?

Did you develop your Labs in your IDE and then CodeLab? How was that experience? When CodeLab rejected your code that seemed to work in your IDE, did you eventually understand what your mistakes were? How did you get past CodeLab rejecting your code? … Query? Friend?

The classes in Lecture/Lab2, were anticipated by first introducing each of them as 1115-like applications (i.e., no objects, just standalone local variables and static method calls within a single class) in Lecture/Lab 1. Did you find this helpful? In particular was the transition over to instance variables and methods, objects, and receivers easier because you were already familiar with the basic details of the application; e.g., was the Container class of Lecture 2 easier to understand because you were familiar with the similar (non-object) version of the container from Lecture 1, or did that parallelism just make matters more confusing?