Section A: Unit by Unit

OOP

Q1:
Which two of the following statements are valid critiques about the design of the A class?

```java
public class A {
    private int foo;
    public String bar;
    private double baz;

    public A(int a, String b) {
        foo = a;
        bar = b;
        baz = Math.random();
    }

    public int getFoo() {
        return foo;
    }

    public void setFoo(int newFoo) {
        if (newFoo > foo) {
            foo = newFoo;
        }
    }
}
```

A. The class features a mixture of public and private instance variables
B. The constructor for A doesn't take in an input for baz.
C. The constructor's assignment operations are missing this, which would cause the program to behave incorrectly.
D. baz is never used inside the class and is inaccessible outside of the class, making it useless.

Given one of the critiques you selected above, briefly explain (at most ~15 words) how you would fix this problem.

*For A: make all of them private (or all of them public) for consistency. For D: write a getter or setter for baz. Or, just delete it!*

Q2:
```java
public class Counter {
    private int count;
```
public Counter() {
    count = 0;
}

public void incr() {
    count++;
}

public int show() {
    return count;
}

public void reset() {
    count = 0;
}

public static void main(String[] args) {
    Counter c = new Counter();
    c.incr();
    c.incr();
    c.incr();

    System.out.println(count);
}

Would the main method compile as written?  A. Yes    B. No

Which of the methods written is functionally a "getter" (accessor) method? A. public Counter() B. public void incr() C. public int show() D. public void reset()

Which one of the following is an invariant of the Counter class? (Remember that an invariant is a statement that is always true about the state of the object.)

A. For a given Counter object, we know that count is always exactly equal to the number of times that incr() has been called.

B. The value of count can only increase after initialization and it will never decrease.

C. The value of count can only decrease when the user calls the incr() method with a negative input.

D. For a given Counter object, we know that count is always less than or equal to the number of times that incr() has been called.

Reference Types
Q1:

```java
public class Greetable {
    private String name;

    public Greetable(String name) {
        this.name = name;
    }

    public void setName(String newName) {
        this.name = newName;
    }

    public void greet() {
        System.out.println("Hi, I'm " + name);
    }

    public static void main(String[] args) {
        Greetable pilar = new Greetable("Pilar");
        Greetable anselmo = new Greetable("Anselmo");

        anselmo = pilar;
        anselmo.greet();
        pilar.setName("Maria");
        anselmo.greet();
        pilar.setName("Pablo");
        pilar.greet();
        pilar = new Greetable("Jordan");
        anselmo.greet();
    }
}
```

Fill in the blanks to represent what gets printed when we run `java Greetable`.

```
Hi, I'm ________
Hi, I'm ________
Hi, I'm ________
Hi, I'm ________
```

**Answer:**

```
Hi, I'm Pilar
Hi, I'm Maria
Hi, I'm Pablo
Hi, I'm Pablo
```

Q2:
Inspect the following class and write a reasonable and meaningful implementation of the `equals()` method. There are several valid answers, but at a minimum your answer should behave differently from using `==` ("double equals") to compare two objects.

```java
public class B {
    private int uniqueID; // guaranteed to uniquely identify a "B"
    private String data;  // multiple "B" objects could have the same data

    public B(String data) {
        uniqueID = generateUniqueID(); // implementation omitted, but assume it exists!
        this.data = data;
    }

    public boolean equals(B other) {
        // your implementation here!
    }
}
```

**Answer:** A correct answer needs to compare, at minimum, one of `data` or `uniqueID` between the two objects. Keep in mind that `data` should be compared using `.equals()` while `uniqueID` must be compared using `==`. A `null` check is optional.

```java
public boolean equals(B other) {
    return other != null &&
            this.uniqueID == other.uniqueID &&
            this.data.equals(other.data);
}
```

### Abstract Data Types

**Q1:**
Given an interface `Drink` and a class `Soda` that implements the `Drink` interface, select all of the lines below that would compile when written in `main` of `Soda.java`.

1. `Drink d = new Soda();`
2. `Drink d = new Drink();`
3. `Soda s = new Soda();`
4. `Soda s = new Drink();`

**Answer:** 1 & 3

Given that `Soda` and `Drink` both compile, that `Soda` implements `Drink`, and that `Soda.java` contains a method with signature `public void open()`.
True or False: We can assume that Drink.java must contain an abstract method with the same signature (public void open()).

List

For the "appendix"

<table>
<thead>
<tr>
<th>method signature</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>get(int i)</td>
<td>return the value at position i in the List.</td>
</tr>
<tr>
<td>set(int i, E e)</td>
<td>set the value at position i in the List to be e.</td>
</tr>
<tr>
<td>size()</td>
<td>return the number of values stored in the List.</td>
</tr>
<tr>
<td>remove(int i)</td>
<td>remove and return the value stored at position i in the List.</td>
</tr>
<tr>
<td>add(E e)</td>
<td>insert the value e at the end of the List.</td>
</tr>
<tr>
<td>add(int i, E e)</td>
<td>insert the value e at position i in the List. i must be between 0 and size().</td>
</tr>
</tbody>
</table>

Q1:
Inspect the following function:

```java
public static int mystery(List<Double> ds) {
    int x = 0;
    for (int i = ds.size() - 1; i >= 0; i--)
        if (ds.get(i) < 0) {
            ds.remove(i);
            x++;
        }
    return x;
}
```

What does the above function do to the input list? Answer: Remove all negative numbers from a given list.

What does the above function's return value represent? Answer: The number of negative values removed from the list.

Q2:
Read the following function that creates a copy of an array A that repeats itself k times and returns this copy:

```java
public static String[] loopKTimes(String[] A, int k) {
    String[] output = new String[A.length * k];
    for (int i = 0; i < A.length; i++) {
        for (int j = 0; j < k; j++) {
            output[i + j * A.length] = A[i];
        }
    }
    return output;
}
```

(For example, `loopKTimes(new String[] {"hi", "bye"}, 3)` returns `"hi", "bye", "hi", "bye", "hi", "bye"`)  

Fill in the following function stub so that the function mimics the behavior of `loopKTimes` but takes in and returns an ArrayList instead. Remember that ArrayLists are dynamically resizing.

```java
public static ArrayList<String> loopKTimes(ArrayList<String> A, int k) {
    ArrayList<String> output = new ArrayList<String>();
    // Place your code here

    return output;
}
```

ANSWER:

```java
public static ArrayList<String> loopKTimes(ArrayList<String> A, int k) {
    ArrayList<String> output = new ArrayList<String>();

    for (int j = 0; j < k; j++) {
        for (int i = 0; i < A.size(); i++) {
            output.add(A.get(i));
        }
    }

    return output;
}
```
/**
 * This node class will store int values.
 */

public class Node {

    // public instance variables
    public int data;
    public Node next;

    public Node(int data, Node next) {
        this.data = data;
        this.next = next;
    }
}

Q1:
Given the following state of nodes:

Node a

rearrange the nodes so that it looks like:

Node a
NOTE:
you are not allowed to access or modify the value in any nodes data instance variable, and you are not allowed to allocate a new node (e.g. the code you write should not contain the `new Node(..., ...)`) in it.
You are allowed to create Node reference variables with something like: `Node temp = a`

```java
public static void foo() {
    // start:
    // +-----+-----+-----+-----+-----+-----+
    // a | 4 | 8 | 1 | 2 | / |
    // +-----+-----+-----+-----+-----+-----+
    Node a = new Node(4, null);
    a.next = new Node(8, null);
    a.next.next = new Node(1, null);
    a.next.next.next = new Node(2, null);

    // end:
    // +-----+-----+-----+-----+-----+-----+
    // a | 2 | 8 | 1 | 4 | / |
    // +-----+-----+-----+-----+-----+-----+

    // TODO: put your code here:
    Node four = a;
    Node eight = a.next;
    Node one = a.next.next;
    Node two = a.next.next.next;

    a = two;
    two.next = eight;
    one.next = four;
    four.next = null;
}
```

Q2:
Complete the following function:

```java
/* Given a reference to the start of a chain of linked nodes
 * and a integer we are searching for
 */
```
* return true if the target is found in the chain,  
* and false if it is not found.
*/

```java
public static boolean search(Node head, int target) {
    // TODO
}
```

## Sorting

For both questions below, consider the following (broken) implementation of selection sort

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>public static void selectionSort(String[] array) {</td>
</tr>
<tr>
<td>1</td>
<td>for (int i = 0; i &lt; array.length - 1; i++) {</td>
</tr>
<tr>
<td>2</td>
<td>int indexOfSmallest = i;</td>
</tr>
<tr>
<td>3</td>
<td>for (int j = 0; j &lt; array.length; j++) {</td>
</tr>
<tr>
<td>4</td>
<td>if (array[indexOfSmallest].compareTo(array[j]) &lt; 0) {</td>
</tr>
<tr>
<td>5</td>
<td>indexOfSmallest = j;</td>
</tr>
<tr>
<td>6</td>
<td>}</td>
</tr>
<tr>
<td>7</td>
<td>}</td>
</tr>
<tr>
<td>8</td>
<td>String temp = array[indexOfSmallest];</td>
</tr>
<tr>
<td>9</td>
<td>array[indexOfSmallest] = array[i];</td>
</tr>
<tr>
<td>10</td>
<td>array[i] = temp;</td>
</tr>
<tr>
<td>11</td>
<td>}</td>
</tr>
<tr>
<td>12</td>
<td>}</td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>}</td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

As mentioned, the code above does not work (though, it still compiles).

### Q1.

Consider we had the following array:

```java
String[] names = {"Michelle", "Lena", "Davis", "Becca"};
```

What if we were to pass this array into the function (e.g. `selectionSort(names)`), what would the `names` array look like after `selectionSort(names)` returns?
**Hint:** if it was properly sorting, `names` would look like:
{"Becca", "Davis", "Lena", "Michelle"}
The code shown above will NOT leave `names` in sorted order

Q2.
Let's try to fix the `selectionSort` code above so that it works properly. For each line that needs to be changed, list its line number and then what the code should be changed to be. You do not have to use all of the rows below:

<table>
<thead>
<tr>
<th>Line Number</th>
<th>Revised Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 2D Arrays

Q1:
Fill in the contents of the 2D array as initialized by this code here:

```java
int[][] output = new int[2][3];
for (int row = 0; row < output.length; row++) {
    for (int col = 0; col < output.length; col++) {
        if (row <= col) {
            output[row][col] = row + col;
        } else {
            output[row][col] = row - col;
        }
    }
}
```

<table>
<thead>
<tr>
<th>output</th>
<th>col</th>
<th>col</th>
<th>col</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Comparing Objects

Q1:
For the following questions, consider the following Treasure class:

```java
public class Treasure {
    public int value;
    public Treasure(int value) {
        this.value = value;
    }

    /* Other methods omitted */
}
```

While there are many valid implementations of a compareTo() method for Treasure, any valid implementation should have the following properties:

1. The method should compile!
2. `x.compareTo(y)` should return a negative number if `x` has a value less than `y`'s value.
3. `x.compareTo(y)` should return a positive number if `x` has a value greater than `y`'s value.
4. `x.compareTo(y)` should return 0 if `x` and `y` have the same value.

For each of the following compareTo() methods, decide if it is valid or invalid. If it is invalid, explain why with reference to the above properties.

Option 1:

```java
public int compareTo(Treasure o) {
    if (this == o) {
        return 0;
    } else if (this < o) {
        return -1;
```


Option 2:

```java
public int compareTo(Treasure o) {
    if (this.value == o.value) {
        return 0;
    } else if (this.value < o.value) {
        return 1 - (int) (Math.random() * 10);
    } else {
        return 1 + (int) (Math.random() * 10);
    }
}
```

Option 3:

```java
public int compareTo(Treasure o) {
    return o.value - this.value;
}
```

Answers: Option 1 is not valid because it does not compile (this and o can’t be compared with <). Option 2 is valid. Option 3 is not valid because it does not return a negative number if this.value is less than o.value.

**Writing Files**

**Q1:**

Here’s a function that takes a 2D array and prints it out with a space between each entry and a new line after each row.

```java
public static void print2DArray(int[][] A) {
    for (int row = 0; row < A.length; row++) {
        for (int col = 0; col < A[row].length; col++) {
            System.out.print(A[row][col] + " ");
        }
    }
    System.out.println();
}
```

Write a function that takes a 2D array of integers and writes it to a file. The file should have a space between each entry and a new line after each row. The function should not return anything.

```java
public static void write2DArray(int[][] A, String filename) {
    OutputStream outStream = new FileOutputStream(filename);
```
// your code here

outStream.close();
}

Answer:

```java
public static void write2DArray(int[][] A, String filename) {
    Out outStream = new Out(filename);
    for (int row = 0; row < A.length; row++) {
        for (int col = 0; col < A[row].length; col++) {
            outStream.print(A[row][col] + " ");
        }
        outStream.println();
    }
    outStream.close();
}
```

Directories:

For the “appendix”?

<table>
<thead>
<tr>
<th>method signature</th>
<th>purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>File(String path)</td>
<td>Constructor, takes in a path to a file to the file to represent.</td>
</tr>
<tr>
<td>isFile()</td>
<td>Method that tests if the File called on is a regular file, true if it is, false otherwise</td>
</tr>
<tr>
<td>isDirectory()</td>
<td>Method that tests if the File called on is a directory, true if it is, false otherwise</td>
</tr>
<tr>
<td>listFiles()</td>
<td>Method that returns a File[]. If the File the method is called on is a directory, the array contains all files in the directory. If the File the method is called on is not a directory, it returns null.</td>
</tr>
</tbody>
</table>

Q1:

Complete the following function:

```java
/* Given a path, determine if that path specifies a parent directory
 * A parent directory satisfies two conditions:
 * - it is a valid directory
 * - it contains at least one directory inside of it
 */
```
public static boolean isParentDirectory(String path) {
    // TODO: Your Code Here
    File f = new File(path);
}

Map
Throughout this course, we have been storing data in arrays and ArrayLists. In both cases, we index into the data structure using an integer. In other words, we can access the elements in those containers by specifying an integer index that uniquely identifies a value in the array or ArrayList.

The goal of the following question will be for you to write a class that allows you to index data using a String as a key instead of an integer. Computer scientists refer to this data structure as a Map, so that's what we'll call the class.

The Map is implemented using an ArrayList<Match>; the Match interface, along with an implementing class SimpleMatch, is provided for you. The Match interface has three methods: getKey(), getValue(), and setValue(). To allow the indexing to be well-defined, we will enforce the constraint that no two Match objects contained in the ArrayList<Match> can have the same key.

Part 1:
Implement the Map constructor. This function takes a filename as input and reads in a list of key to value matches from the file. Note that the key is of type String, and the values are of type double: remember your readXYZ() methods from In.java! The file format is as follows, where N refers to the number of Matches represented in the file:

N
key1 value1
key2 value2
key3 value3
...
keyN valueN

You can assume that the file will be formatted correctly and that it will contain no duplicate keys. The constructor should read in the file and store the key to value matches in the ArrayList<Match> instance variable called matches.

    public Map(String inputFile) {
        this.matches = new ArrayList<Match>();
        // your code here
Part 2:

Next, write two tests for the provided `get()` method.

The first test should create a `Map` object from the file `test1.txt` and then call the `get()` method on the `Map` object with the key "Aditya". The test should then check that the value returned by the `get()` method is equal to 63.4 (representing the number of hours he spent in Office Hours this semester!)

```
Part1.txt
3
Aditya 63.4
Bhrajit 94.5
Sukya 192.3

@Test
public void getElementPresent() {
    // your code here
}
```

The second test should create a `Map` object from the file `test1.txt` and then call the `get()` method on the `Map` object with a key that is not present in the file. The test should then check that the value returned by the `get()` method is `Double.NEGATIVE_INFINITY`.

```
@Test
public void getElementNotPresent() {
    // your code here
}
```

Part 3:

Next, implement the `insert()` method. This method takes a key and a value as input and updates our `Map` to match the key with that value. If a `Match` object with the given key already exists in the `ArrayList<Match>`, then the method should update the existing `Match object with that key` and return `false` to indicate that the size of the `Map` has not changed. Otherwise, the method should create a new `Match` object, insert it into `matches`, and return `true`. We have provided two test cases that further demonstrate the intended behavior.

```
@Test
public void insertKeyNotPresent() {
    Map m = new Map("test1.txt");
    boolean result = m.insert("key4", 3423);
    assertTrue(result);
    assertEquals(3423, m.get("key4"), 0.01);
    assertEquals(m.size(), 4);
}
```
@Test
def insertKeyAlreadyPresent() {
    Map m = new Map("test1.txt");  // has a match for "Aditya"
    boolean result = m.insert("Aditya", 100.3);
    assertFalse(result);
    assertEquals(100.3, m.get("key1"), 0.01); // ignore the delta for grading
    assertEquals(m.size(), 3);
}

def insert(key, value) {
    // your code here
}

class Map()
{
    private ArrayList<Match> matches;

    def Map(String input) {
        matches = new ArrayList<Match>();
        In inStream = new In(input);  
        int numMatches = inStream.readInt();
        for (int i = 0; i < numMatches; i++) {
            String key = inStream.readString();  
            double value = inStream.readDouble();
            Match m = new SimpleMatch(key, value);
            matches.add(m);
        }
    }

    def get(key) {
        for (int i = 0; i < matches.size(); i++) {
            if (m.key.equals(key)) {
                return m.getValue();
            }
        }
        return Double.NEGATIVE_INFINITY;
    }

    def insert(key, value) {
        for (int i = 0; i < matches.size(); i++) {
            if (m.key.equals(key)) {
                m.setValue(value);
                return false;
            }
        }
    }
}
```java
    }
    }
    Match m = new SimpleMatch(key, value);
    matches.add(m);
    return true;
    }
	public boolean delete(String key) {
		for (int i = 0; i < matches.size(); i++) {
			if (m.key.equals(key)) {
				matches.remove(i);
				return true;
			}
		}
		return false;
	}
	public int size() {
		return matches.size();
	}
}

class SimpleMatch implements Match {
    private String key;
    private double value;

    public SimpleMatch(String key, double value) {
        this.key = key;
        this.value = value;
    }

    /* Other methods omitted for space; they behave as the method names suggest for getters and setters. */
}