CIS 110 Fall 2020 Midterm 2 Questions

Note: This exam had multiple versions. We have selected a subset of those versions to simulate the length of the exam. The questions we have redacted covered no different material. However, the T/F section had a random subset given to students, but we are leaving all the questions in here for extra practice. Therefore this practice exam is slightly longer than the actual administered exam.

True/False

1a. You cannot overload constructors in Java

1b. The following code prints Shivin

```
public class Student {
	private String name;
	public Student(String name) {
		name = name;
	}

	public static void main(String[] args) {
		Student shivin = new Student("Shivin");
		System.out.println(shivin.name);
	}
}
```

1c. The following code prints a memory address

```
public class Student {
	private String name;
	public Student(String name) {
		name = name;
	}

	public String toString() {
		return name;
	}
	public static void main(String[] args) {
		Student shivin = new Student("Shivin");
		System.out.println(shivin);
	}
}
```

1d. The following code compiles

```
public class Student {
	private String name;
	private Student[] friends;
```

```
      public Student(String student) {
            name = student;
            friends[0] = this;
      }
}
```

1e. The following code correctly removes the first Node in a Linked List.

```
public void removeFirstNode() {
  headNode.next = headNode;
}
```

1f. The following code snippet adds a node after the first node

```
public void mystery(int value)
    {
        Node newNode = new Node(value);
        newNode.next = head;
        head = newNode;
    }
```

1g. This code snippet prints “j”

```
LinkedList sandwich = new LinkedList ();
sandwich.append("j");
sandwich.append("pb");
sandwich.insert(1, "j");
sandwich.insert(1, "pb");
sandwich.remove(1);
System.out.print(test.get(2));
```

1h. ArrayLists and LinkedLists are dynamically sized

1i. The above template for the list Gifts correctly implements the GiftList interface

```
public interface GiftList {
    public void add(Gift a);
    public void gift(Gift b);
}

public class Gifts implements GiftList {
   public void addGift(Gift a) {
      *code to add a present to the list*
   }
   public void giveGift(Gift b) {
       *code to gift a present from the list*
   }
}
```

1j. You cannot add an object to a list at a position greater than the size of the list

1k. The below tests display the same results

```
public void testIsMammal() {
String animal = “cat”;
boolean expected = true;
boolean actual = isMammal(animal);
assertEquals(expected, actual);
}
```

And

```
public void testIsMammal() {
Boolean actual = isMammal(“cat”);
assertTrue(actual);
}
```

1l. You write a test case for a function called countAttendance. The test case fails. True or false: there must be a bug in countAttendance.

1m. If an IllegalArgumentException is thrown, this test under this header would pass

```
@Test (expected = IllegalArgumentException.class)
```

1n. This is a file called Buzz.java

```
public class Buzz {
      private int fizz;
      public Buzz() {
            fizz = int (Math.random() * 10);
      }
}
```

The following lines will cause a compilation issue when written in a separate class `TestBuzz.java`.

```
Buzz b = new Buzz();
System.out.println(b.fizz);
```

1o. This is a file called `Buzz.java`.

```
public class Buzz {
      private int fizz;
      public Buzz() {
            fizz = int (Math.random() * 10);
      }
}
```

The following lines will cause a compilation issue when written in a separate class `TestBuzz.java`.

```
Buzz b = new Buzz();
System.out.println(b);
```

1p. If `a.equals(b)` is `true`, then `a == b` is `true` always.

1q. If `a == b` is `true`, then `a.equals(b)` is `true` always.

1r. If a class `Novel` has a method `public static void printSummary()`, and if book is an instance of class `Novel`, then calling `book.printSummary()` will cause an error.

1s. If TV is an interface, the following code will compile: `TV tv = new TV();`

1t. An interface must be implemented by some class to compile

1u. If Grape & Blueberry implement a Fruit interface, we can do the following

```
Fruit g = new Grape();
Fruit b = new Blueberry();
Fruit[] fruits = {g, b};
```

Short Fill in the blank

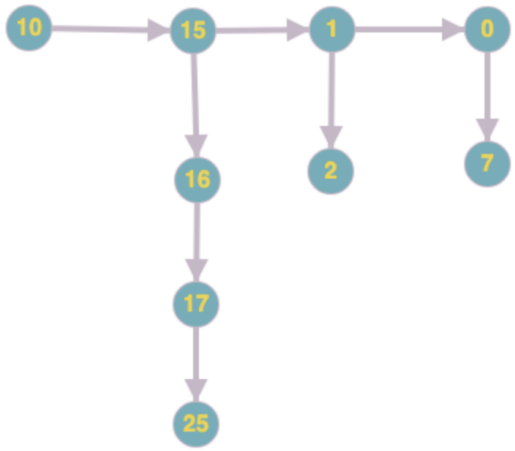
1. A _____ field is associated with the class rather than any object. That means that there is only one instance of that field in memory.
2. If the visibility modifier of a field is _____, then accessing its value outside of its class definition requires the use of a getter.
3. The _____ is a special method of a class that creates a new object from that class. It is easy to identify because it shares a name with the class itself.
4. Math.random() and PennDraw.advance() are _____ functions, which explains why each function can be invoked without creating an instance of their respective classes.

Fill in the blank Functions

Problem 1 - Linked Lists

Version 1: Holiday Party

3a: The TAs are throwing a holiday party and are stringing up lights to celebrate. As they removed each bulb from the box, they gave it a random brightness value. They then connected the lights randomly to each other, either on the bottom of or to the right of each previous light. They need your help extending the chain of lights by providing an array that lists the lights' brightness value from left to right, first going down as far as possible for each bulb. For example, you should return [10, 15, 16, 17, 25, 1, 2, 0, 7] from the diagram below.



Assume that the function is initially called with `linkTheLights(head, 0, arr);`
Where head is the top-left node and arr is an integer array with size of the number of lights. Also assume that only bulbs in the top row (10, 15, 1, and 0) can point to another bulb to its right. For example, bulb 16 would never be able to point to a bulb to its right. Do not worry about the syntax of the PartyLight class (i.e. public static class PartyLight). Even though you should not worry about this syntax, **please do not change it for compilation purposes.**

```
public static class PartyLight {
    PartyLight right; //pointer to right PartyLight
    PartyLight down; //pointer to down partyLight
    int brightness; //value of light's brightness

    //constructor with right and down pointers + brightness parameter
    PartyLight(PartyLight r, PartyLight d, int v){
        this.right = r;
        this.down = d;
        this.brightness = v;
    }
```

```
    //constructor with just brightness parameter
    PartyLight(int v) {
        this.brightness = v;
    }
}
```

```
public static void linkTheLights(PartyLight curr, int index, int[] arr){
    if(___(1)___){
        return;
    }
    arr[index] = ___(2)___;
    index++;

    PartyLight n = curr;

    while(___(3)___){
        n = ___(4)___;
        arr[index] = ___(5)___;
        index++;
    }

    linkTheLights(___(6a)___, ___(6b)___, ___(6c)___);

}
```

```
public static void linkTheLights(PartyLight curr, int index, int[] arr){
    if(_______){ //#1 curr == null
        return;
    }
    arr[index] = ____________; //#2 curr.brightness
    index++;

    PartyLight n = curr;

    while(_________){ //#3 n.down != null
        n = _____; //#4 n.down
        arr[index] = _________; //#5 n.brightness
        index++;
    }

    linkTheLights(_______________);
    // #6a curr.right, #6b index, #6c arr
}
```

Problem 2 - 2D arrays

Version 2: Schedule Filler

4b: Michelle is trying to set up her schedule for next semester. She is representing her schedule as a 2D array of integers, with each row representing a day. Each int in a row represents an hour: 0 if she has no class then or a positive course number if she has class. However, when she tried to download her schedule, it only included the beginning and end of each class! Help Michelle write a function to fill in the gaps in her schedule with the proper course numbers.

The input comes in the following format:

```
int[][] schedule = {
        {0, 1, 0, 1, 0},
        {0, 14,  0,  0, 0, 0, 14},
        {0, 0, 1, 0, 0, 1},
        {0, 0, 3, 0}
        {0, 0, 44, 0, 44}
}
```

Each row in our schedule contains either 1 positive integer or 2 positive identical integers representing courses. She only has one class in a day. If there is only one integer in the row, we do not have to change anything. If there are 2 integers however, then we have to fill in the space between the 2 with the same integer value. Thus, the output looks like this:

```
int[][] schedule = {
        {0, 1, 1, 1, 0},
        {0, 14,  14,  14, 14, 14, 14},
        {0, 0, 1,1, 1, 1},
        {0, 0, 3, 0}
        {0, 0, 44, 44, 44}
}
```

Notice that the rows are not of the same length: the input may be a ragged matrix. Fill in the blanks of the following function:

```
public static void scheduleFiller(int[][] schedule) {
      for (int row = 0; row < schedule.length; row++) {
            int numCounter = 0;
            int class = 0;
            for (int col = 0; _____; col++) {
                  if (schedule[row][col] > ___) {
```

```
                        numCounter++;
                        class = schedule[row][col];
                  }
            }
            if (________) {
                  continue;
            }
            boolean inClass = false;
            for (int col = 0; col < schedule[row].length; col++) {
                  if (______ && !inClass) {
                        inClass = true;
                  } else if (schedule[row][col] == 0 && inClass) {
                        __________ = class;
                  } else if _____________ {
                        break;
                  }
            }
      }
}
```

Long Coding

Problem 2 - Feeding Time

5b: During quarantine, Gian adopted dozens of cats to keep him company. He has to feed these cats 2 servings of food every night. He has both salmon and tuna food at his disposal. To prevent a feeding frenzy, Gian has trained his cats to line up in two queues, one for each type of food, each modeled as a List.

Each Cat must receive 2 servings of food - this can be both salmon, both tuna or 1 salmon + 1 tuna (the fish doesn't matter as long as they get 2 servings). With both queues, Gian gives out only 1 serving at a time. At a given point of time, a Cat can be in both queues **but cannot appear multiple times in the same queue**. **And at no point should a Cat who has already received 2 servings be in a queue.**

Here are the List interface methods you may use for a List where Cat is the object type the List is storing. We've covered these in class, so it is expected you know what they do. You don't need to worry about import statements.

```
boolean add(Cat c);
void add(int index, Cat c);
boolean remove(Cat c);
boolean contains(Cat c);
boolean isEmpty();
Cat get(int index);
int size();
```

We have also given you the Cat API below. You will find the functions in this class helpful.

```
Cat {

    /***
    * constructor which takes in an animal ID number and
    * initializes their servings of food to be 0
    **/
   Cat(int id);

   /***
    * method which increases the number of servings
    * a Cat has received
    **/
    void feed();

   /***
    * getter method for number of servings
    * a Cat has received
    **/
   int getNumServings();

   /***
    * determines if two Cat objects are equal
    * (every Cat has a unique animal ID identifier)
    **/
   boolean equals(Object o);
}
```

Before implementing the FeedingQueues class, we will write a few test cases. We highly recommend you paste the following code into your Codio environment, and then copy and paste the code for both classes in the essay textbox following this question.

Testing

The fields in FeedingQueues have been made public for testing convenience. You do NOT need to test for *unintended* side effects, just what is specified in the function headers. A test for the constructor has been completed for you as an example. Please write tests for the following cases and name them appropriately:

- A test for joinSalmon() where the input Cat is already in the salmon queue
- A test for joinSalmon() where the input Cat is not already in the salmon queue and has received fewer than 2 servings
- A test for feedSalmon() where the Cat at the head of the queue should be receiving their second serving

```
import java.util.*;
import static org.junit.Assert.*;
import org.junit.*;

public class FeedingQueuesTest {

    @Test
    public void constructorTest() {
        FeedingQueues q = new FeedingQueues();
        assertTrue(q.salmon.isEmpty());
        assertTrue(q.tuna.isEmpty());
        assertEquals(0, q.totalFed);
    }



}
```

Implementation

Once you have completed your tests, implement the FeedingQueues class as described. (Note: we will not ask you to implement joinTuna() and feedTuna() as they are symmetric to the salmon functions).

```
import java.util.*;

public class FeedingQueues {
    public int totalFed; // total number of cats fed
    public List salmon; // the salmon queue
    public List tuna; // the Tuna queue

    /* Both Lists should be initially empty and the total number of cats
fed should be 0.
    */
    public FeedingQueues() {
        //TODO
    }

    /* Add the Cat c to the end of the salmon queue. However, you must
throw an IllegalArgumentException with the relevant error messages if the
Cat c has already received 2 servings of food or if the Cat c is already in
the salmon queue. You can assume that Cat c is not null.
    */
    public void joinSalmon(Cat c) {
        //TODO
    }

/*
You should remove the Cat at the front of the salmon queue (which is index
0) & "feed" them (see the feed() function in the Cat class). After getting
fed, if that Cat has already received 2 servings, remove them from the Tuna
queue as well (if they are in it) and update the total number of cats fed.
If that Cat still hasn't received the 2 servings, you should add them back
to the end of the salmon queue. If there is no one in the queue, you should
do nothing.
*/
    public void feedSalmon() {
        //TODO
    }
}
```

[redacted]