

Directions: Give yourself 50 minutes to work through this “exam”. Use a quiet space where you can concentrate.

Note: This midterm is **less than half as long** as the actual thing. It’s intended as practice only. No guarantee that it matches up with the real deal at all- this is totally unofficial.

Some problems are more in depth than you are expected to know (especially the first one).

Problems by Jonathan Shewchuck.

Problem 1: Box and pointer

Observe the following class:

```
public class SListNode {  
    public Object item;  
    public SListNode next;  
}
```

(4 points) Consider the following constructor in an `SListNode` class that stores ints. Suppose the main method calls `new SListNode(5)`. **Draw the stack and the heap** just before the **deepest recursive call** of the following constructor returns.

```
public SListNode(int i) {  
    item = i;  
    if (i > 1) {  
        next = new SListNode(i / 3);  
    } else {  
        // Draw the stack and heap at this moment.  
    }  
}
```

Note: Frames go on the stack. Objects go on the heap. This is the same style as the java visualizer from class.

Frames Here! (Stack)

Objects Here! (Heap)

Problem 2: Dating Profile

Fill the blanks in so the code compile and runs without errors or an infinite loop. Classes are not in the same package.

What is printed when you run Profile.main() ?

```
public interface X {
    public void whatever();
}

public abstract class Profile implements X {
    _____ double weight;           // Maximize my
    privacy, please.

    public Profile(double weight) {
        this.weight = weight;
    }

    public _____ double date(double food);

    public int date(int food) {
        return _____ date(_____ food);
    }

    public String introduce() {
        return "I'm neurotic and vindictive and I weigh " + weight;
    }

    public static void main(String[] args) {
        Profile p = new _____(260.0);
        _____ .diet(p.weight, 25.0);
        System.out.println(p.introduce());
    }
}

public class DatingProfile extends Profile {
    public DatingProfile(double weight) {
        _____(weight - 70.0);
    }

    public _____ date(_____ food) {
        weight = weight + (double) food;
        return weight;
    }
}
```

```
public void diet(double weight, double loss) {  
    weight = weight - loss;  
}  
  
public String introduce() {  
    return "I'm feisty and spontaneous and I weigh " + (weight -  
60);  
}  
  
_____
```

Output:

Scratch work goes here

Problem 3: Lengthening Runs

In a list of ints, a *run* is a subsequence of ints that are all the same. For instance, there are three runs in the list 4 4 4 4 1 7 7 7: a run of 4's, a run of 1's (with just one member), and a run of 7's.

Implement the method `SListNode.lengthenRuns` below, which **increases the length of each run by one item, then returns the number of runs**. For example, if the list is initially 4 4 4 4 1 7 7 7, then it should be 4 4 4 4 4 1 1 7 7 7 7 when `lengthenRuns` is finished, and `lengthenRuns` should return 3. Note that there is no `SList` class; only listnodes. Modify the list whose head is this. If you call any method besides the `SListNode` constructor provided, you must provide the implementation here.

```
public class SListNode {
    protected int item;
    protected SListNode next;

    SListNode(int i, SListNode n) {
        item = i;
        next = n;
    }

    public int lengthenRuns() { // Implement me!
```