CS 61B
Spring 2017

Data Structures
Practice Midterm 1 Solutions

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 `int`s. 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.

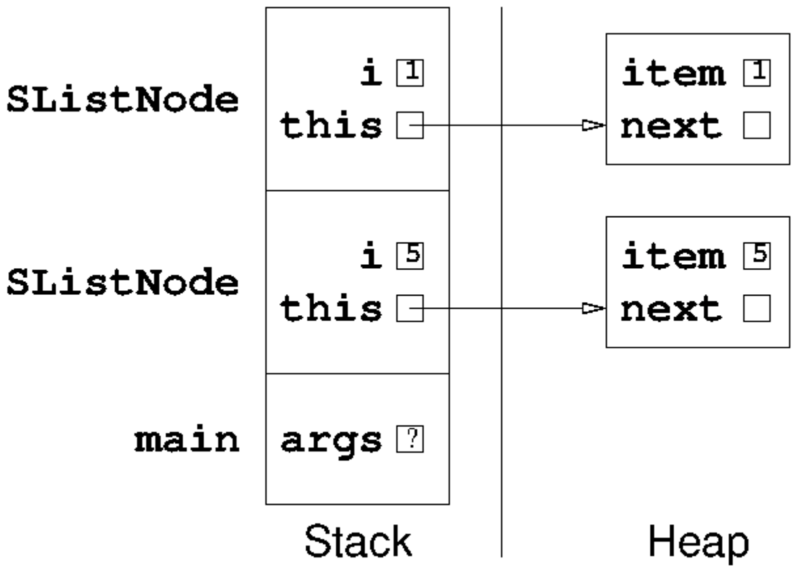
```
  public SListNode(int i) {
    item = i;
    if (i > 1) {
      next = new SListNode(i / 3);
    } else {
      // Draw the stack and heap at this moment.
    }
  }
```

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)

Solution:

d.



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

Solution:

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

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

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

  public abstract double date(double food);

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

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

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

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

  public double date(double 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);
  }

  public void whatever() { }
}
```

The code prints: I'm feisty and spontaneous and I weigh 130.0

Problem 3: Lengthening Runs

In a list of `int`s, a *run* is a subsequence of `int`s 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!
```

Solution:

Here's an example of a recursive solution.

```
  public int lengthenRuns() {
    if (next == null) {
      next = new SListNode(item, null);
      return 1;
    }

    if (item == next.item) {
      return next.lengthenRuns();
    }

    next = new SListNode(item, next);
    return 1 + next.next.lengthenRuns();
  }
```

Most students wrote iterative solutions (as opposed to recursive). This has the big advantage that you can process much longer lists without running out of stack space, but the code is somewhat more complicated.

```
  public int lengthenRuns() {
    int count = 1;
    SListNode n = this;
    while (n.next != null) {
      if (n.item != n.next.item) {
        n.next = new SListNode(n.item, n.next);
        n = n.next;
        count++;
      }
      n = n.next;
    }

    n.next = new SListNode(n.item, null);
    return count;
  }
```

One final reminder to the students who took the exam: In Java, this can never be null. Many of you started with comparisons like if (this == null), which are usually harmless but are always completely unnecessary.