University of California at Berkeley
Department of Electrical Engineering and Computer Sciences
Computer Science Division

Autumn 2006 **Jonathan Shewchuk**

CS 61B: Midterm Exam I

This is an open book, open notes exam. Electronic devices are forbidden on your person, including cell phones, iPods, headphones, and PDAs. Turn your cell phone off and leave all electronics, except your laptop, with the instructor, or risk getting a zero on the exam. **Do not open your exam until you are told to do so!**

Name:_____

Login:_____

Lab TA:_____

Lab day and time:_____

Do not write in these boxes.

Problem #	**Possible**	**Score**
1. Java	6	
2. Inheritance	7	
3. Heap and stack	7	
4. Array to list	5	
Total	25	

Problem 1. (6 points) **Java bugs.**

After you finished studying late last night, you fell into a fitful sleep and dreamt the following code. Unfortunately, you wrote four buggy lines. Fix them, so that all three methods work correctly. Each fix should involve changing just part of a line of code. The fact that everything is `public` does not count as a bug.

```
public class Quantity {
  public String thing;       // The thing being measured.
  public double amount;      // Its numerical quantity.

  // Constructor.
  public Quantity(String thingString, double amount) {
    thing = thingString;
    amount = amount;
  }

  // Constructor for thing with quantity 100.  Calls the other constructor.
  public Quantity(String thingString) {
    Quantity(thingString, 100.0);
  }

  public static void main(String[] args) {
    Quantity q = Quantity("I love Java this much:  ");
    System.out.println(this.thing + this.amount);
  }
}
```

Problem 2. (7 points) **Inheritance.**

```
public abstract class Cat {
  public void greet() {
    System.out.println("Meow");
  }
}

public class Tomcat extends Cat {
  public void greet(Cat c) {
    System.out.println("What up fool?");
    c.greet();
  }
}
```

```
public class Siamese extends Cat {
  public void greet() {
    System.out.println("Meeeeow!");
  }

  public void greet(Cat c) {
    System.out.println("Hellow");
  }
}
```

What does each of the following code fragments print? If a code fragment causes an error, say whether it is a compile-time error or a run-time error.

a. `(new Tomcat()).greet();`

_____

b. `(new Tomcat()).greet(new Siamese());`

_____

c.
```
Cat c = new Tomcat();
Tomcat t = c;
t.greet(c);
```

_____

d.
```
Cat c = new Siamese();
((Cat) c).greet();
```

_____

e.
```
Cat c = new Siamese();
c.greet(new Tomcat());
```

_____

f.
```
Cat c = new Siamese();
((Tomcat) c).greet(c);
```

_____

g.
```
Cat c = new Cat();
c.greet(c);
```

_____

Problem 3. (7 points) **The heap and the stack.**

The following code creates a linked list representing the Fibonacci series in reverse order. (Don't worry if you don't know what that is.) Suppose we execute `main`. Draw the stack and heap at the moment when the topmost `fibonacci` call on the stack is about to "`return n`". Specifically, draw a box-and-pointer diagram with a box for every stack frame, local variable, object, and field in memory at that moment. Include the stack frames for all methods in progress, and illustrate which entities are inside those stack frames. Don't forget "`this`". Entities on the stack should be on the left-hand side of the page, and entities on the heap should be on the right-hand side.

```
public class ListNode {
  public int item;
  public ListNode next;

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

  public ListNode fibonacci(int i) {    /* Appends i more terms to series. */
    /* Next number is sum of previous two numbers in series. */
    ListNode n = new ListNode(item + next.item, this);
    if (i <= 1) {
      // Draw the stack and the heap when execution reaches this point.
      return n;
    } else {
      return n.fibonacci(i - 1);
    }
  }

  public static void main (String[] args) {
    /* First 2 numbers in series are 1. */
    ListNode n = new ListNode(1, new ListNode(1, null));
    n = n.fibonacci(3);                            /* Append 3 more terms. */
  }
}
```

```
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
                                                  |
Draw stuff on the stack on the left side | Draw stuff on the heap on the right
```

Problem 4. (5 points) **Creating a Doubly-Linked List.**

Write a method called `makeList` in the `DListNode` class below. `makeList` takes an array `counts` of `int`s, constructs a doubly-linked list, and returns the list's **first** `DListNode`. The list is **not** circularly linked and does **not** have a sentinel node. There's **no** `DList` class. In the returned list, the first `counts[0]` items are the number `counts[0]`, the next `counts[1]` items are the number `counts[1]`, etc. For example, if the input array is `[ 3 5 1 ]`, then the output linked list is .

Your solution should manipulate `next` and `prev` pointers directly. If you call any other methods, you must include them here. Assume there are no negative numbers in the array. Do not construct another array.

```
public class DListNode {
  public int item;
  public DListNode prev, next;

  public DListNode(int i, DListNode p, DListNode n) {
    item = i;
    prev = p;
    next = n;
  }

  public static DListNode makeList(int[] counts) {

  }
}
```

Check here if your answer is continued on the back. |---|