

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

Spring 2009

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. Quickies	7	
2. Inheritance	8	
3. Remove node from list	10	
Total	25	

Problem 1. (7 points) **Quickies.**

- a. (1 point) Briefly explain the difference between an instance variable and a class variable.
- b. (1 point) Can you use the `super` keyword in a static method? Explain.
- c. (3 points) What is the output of this program? _____.
Explain why. _____.

```
public class What {
    public long n;

    public void increment() {
        n++;
    }

    public static void reset(What w) {
        w.increment();
        w = new What();
        w.n = 0;
    }

    public static void main(String[] args) {
        What w = new What();
        w.n = 7;
        reset(w);
        System.out.println("The number is " + w.n);
    }
}
```

- d. (2 points) What's wrong with the following code? Specifically, what does this code do? (Yes, it does compile and run.)

```
public class Soda {
    public String name;

    public Soda() {
        Soda pop = new Soda();
        pop.name = "Dr. Pepper";
    }

    public static void main(String[] args) {
        System.out.println((new Soda()).name);
    }
}
```

Problem 2. (8 points) Inheritance.

Fill in the blanks so that the following code compiles and runs without throwing an exception. (The code is all in one file, Ccc.java.) Note that some blanks may require more than one word.

```
_____ java.io.*;

_____ Aaa {
    public _____ number();
}

_____ Bbb {
    public int[][] i;

    public Bbb(int j) {
        i = _____;
        i[3][5] = j;
    }

    public _____ number() {
        return 12.73;
    }

    public _____ void cureCancer();
}

public _____ Ccc _____ Bbb _____ Aaa {
    public Ccc() {
        _____;
    }

    public void cureCancer(int i) {
        _____ .i[1][1] = 4;
    }

    public void _____ () {
        System.out.println(number());
    }

    public static void main(String[] args) {
        Aaa a = _____ ();
        Bbb b = _____ a;
        _____ .i[0][0] = 1;
    }
}
```

Problem 3. (10 points) **Removing a Node from a List.**

- a. (6 points) Write a method called `removeNode` in the `SList` class below, which implements a singly-linked list. `removeNode` takes an `SListNode` `node` which you *know* is in this list, and removes it. (Your method does not need to work, and may even crash, if `node` is not in the list or is `null`.) After you're done, all the other nodes (that you didn't remove) must still be in the list, in the same order as before.

Your solution should manipulate `next` pointers directly. If you call any other methods, you must include them here. There is no `size` field. Your method should be fast.

```
public class SList {                               | public class SListNode {
    public SListNode head;                         |     public Object item;
                                                    |     public SListNode next;
    public void removeNode(SListNode node) {      | }
}
```

```
    }
}
```

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

Part b. is on the next page...

- b. (4 points) Now, write the same method for a doubly-linked list in the `DList` class below, which is a subclass of `SList`. Observe that `DListNode` is a subclass of `SListNode` too. This doubly-linked list class has **no sentinel**; it has both a `head` and a `tail` pointer.

Although the `removeNode` method takes an `SListNode` parameter (so that overriding will work correctly), you should assume that the object passed in is **always** a `DListNode`. To help you out, we've included a cast to give you a more useful parameter `dnode`. (Again, your method does not need to work, and may even crash, if `node` is not in the list, is null, or is not a `DListNode`.)

Your doubly-linked `removeNode` method **must** reuse code by calling the superclass method. That way, you will only need to write code to update the `tail` reference or some node's `prev` reference.

```
public class DListNode extends SListNode
    // inherits Object item and SListNode next
    public DListNode prev;
}

public class DList extends SList {
    // inherits SListNode head
    public DListNode tail;

    public void removeNode(SListNode node) {
        DListNode dnode = (DListNode) node; // Assume this cast always succeeds.
```

```
    }
}
```

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

Don't forget that `next` and `node` have static type `SListNode`, and therefore must be cast to `DListNode` if you want to use their `prev` fields.