CS 61B Discussion 4 EP Solutions Spring 2017

1 What Would Java Print? (Spring 2015 Q1)

In the four blanks beside the print statements, write the result of the print statement. For your reference, the definition of the `IntList` class is given below the code block.

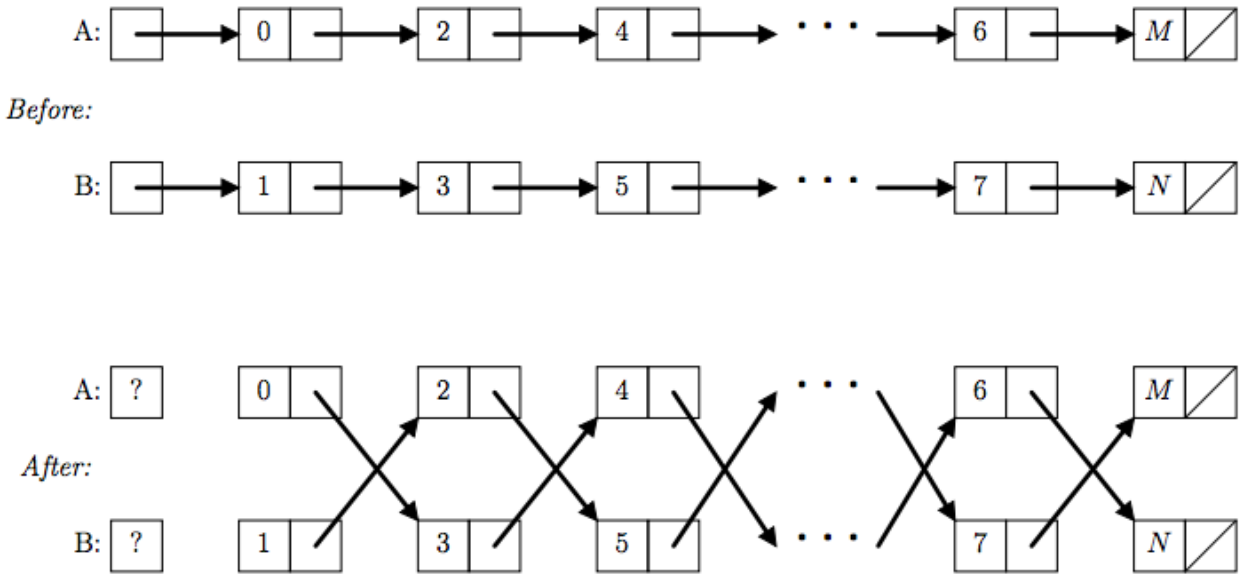
```
public class Problem1 {
    public static void main(String[] args) {
        IntList a = new IntList(5, null);
        System.out.println(a.head);                          ___
        IntList b = new IntList(9, null);
        IntList c = new IntList(1, new IntList(7, b));
        a.tail = c.tail;
        a.tail.tail = b;
        b.tail = c.tail;
        IntList d = new IntList(9002, b.tail.tail);

        System.out.println(d.tail.tail.tail.head);           ___
        System.out.println(a.tail.head);                     ___

        c.tail.tail = c.tail;

        System.out.println(a.tail.tail.tail.tail.head);      ___
    }
}


public class IntList {
    private int head;
    private IntList tail;

    public IntList(int i, IntList n) {
        head = i;
        tail = n;
    }
}
```

First print: 5
Second print: 9
Third print: 7
Fourth print: 7

Some tips: There are exactly five IntList instances created, so for example when `IntList c = new IntList(1, new IntList(7, b))` is called, you shouldn't be creating a copy of `b`. Box and pointer diagrams are highly recommended as you work through the code.

2 Intersecting IntLists (Fall 2014 Q2b)

Here, `A` and `B` are declared to be IntLists, and the two boxes in each object are respectively the head and tail of an IntList. Do not change any of the values in the head fields. Do not create new objects. Assume that all four rows have the same number of objects (so that $N = M + 1$). Fill in the blanks to convert the "Before" diagram into the "After" diagram. **Do not introduce any new variables other than those shown in the diagrams**. Put at most one statement or expression in each blank. You need not use all the blanks.




```
while (A != null) {
    IntList t1 = A.tail;
    A.tail = B.tail;
    B.tail = t1;
    B = A.tail;
    A = t1;
}
```

To understand the solution code, it again helps to draw box and pointer diagrams. Basically what's happening is that on each iteration, the code is able to cross the two pointers, then move the A and B pointers up forward to continue the iterations. The t1 variable is necessary because without it, if we were to reassign A.tail, we would lose access to the rest of the list. One can typically think of temp variables in linked lists problems as keeping track of locations as other pointers are being manipulated.

3 Inheritance (Fall 2015 Q4)

Consider three distinct Java classes, `A`, `B`, and `C`.

(a) Suppose that the following code compiles and runs without any exceptions. What can you say about the relationship between the classes `A`, `B`, and `C`?

```
A x1 = new C();
B x2 = x1;
B x3 = (B) x1;
B x4 = (C) x1;
```

C extends A, and A extends B. Since the first line works, we know that C is a subclass of A. The next line tells us that A is a subclass of B. We can try other combinations, for example A -> C -> B, but that wouldn't work because the first line wouldn't work.

(b) Now suppose the code in part (a) fails to compile, but if we remove the second line it compiles and runs without any exceptions. What can you say about the relationship between the classes `A`, `B`, and `C`?

C extends B, and B extends A. The second line failing and third line working tells us that we have to do an explicit cast to the compiler in order to tell the compiler that we can store x1 into a variable of static type B.

(c) Now suppose that the code in (a) fails to compile, and when we remove the second and fourth lines it compiles, but causes a `ClassCastException` at runtime. What can you say about the relationship between the `A`, `B`, and `C`?

C extends A and B extends A. The ClassCastException at runtime tells us that the third line is thought to be okay by the compiler, but when it is actually ran it turns out that since B and C have no other relationship other than A, there's no way to store a C type variable into a variable of static type B.

B extends C, and C extends A is also a valid solution. Downcasting in Java is typically dangerous. This is because downcasting restricts the type. Thus, when you try to cast C as a B static type, not all C types are B types, and thus a ClassCastException is thrown.

This Stack Overflow thread on casting is very helpful:
`http://stackoverflow.com/questions/5289393/casting-variables-in-java`