Midterm 1 Review Document

CS61B Fall 2016

Antares Chen

Answers!

Introduction

This document is meant to provide you supplementary practice questions for the upcoming midterm. It reflects all material that you will have already seen in labs and lecture. Do not use this as a "be all end all" guide! It is still highly recommended that you review previous and external course material.

For example, I suggest that you do many MANY practice midterm 1s from previous semesters. Use the midterms a heuristic for your knowledge of the material, then use the lab guides and textbook to relearn the material.

Finally, make sure you don't stress! This class is hard and will only be made harder if you stress yourself out. But you smart, you loyal #blessup. Ask lots of questions and let us the TA's help you (believe me when I say we want you guys to succeed).

If you find yourself beginning to panic, simply pause, breathe, and believe. In the off chance you don't believe in yourself, then believe in me who believes in you.

Organization

- This document will be split into four parts:
    - Know Your Java: practice on object oriented design, static vs dynamic type, and the java language in general
    - Arrays: practice on using arrays
    - LinkedLists: practice on using linked lists
- The last three parts are further divided into three difficulties
    - Easy-mode: basic understanding of the concept. These are to make sure you know the syntax of the construct.
    - Medium-mode: expected understanding. This should be the level you are at after doing the labs, readings, and quizzes
    - Hard-mode: these are non-trivial exercises that are obfuscated and meant to be difficult. If you can seamlessly complete these then feel good about yourself!

(Know Your Java) This Code Sucks!

This code sucks… no seriously some of the below code examples are broken. For this question, we will be looking at the following classes.

```
public class A {
    private int valA = 2;
    public void f() {
        this.g();
    }
    public void g() {
        System.out.println("A:" + valA);
    }
    public int h() {
        return valA;
    }
    public static A createA() {
        return new A();
    }
}
```

```
public class C extends B {
    private int valC = 42;
    public void f() {
        this.g();
    }
}
```

```
public class B extends A {
    private int valB = 15;
    protected int value = 1337;
    public void g() {
        System.out.println("h:" + h() + " z:" + valB);
    }
    public void banana() {
        System.out.println("no");
    }
}
```

Beneath are a number of coding examples. Next to each, determine if the code can run. If it can, write down the output, else state what type of error (compile-time vs runtime) is thrown and for what reason.

```
// to be placed in class A
public static void main(String[] args) {
    A this = new A();
    this.g();
}
```

compile time!

can't reassign this

`// to be placed in class A`
`public static void main(String[] args) {`
`    A thing = this.createA();`
`    thing.f();`
`}`
	createA() is a static method! shouldn't access it like an instance var.

`// to be placed in class A`
`// what is this?`
`public A(int valA) {`
`    this.valA = valA;`
`}`
	This is a constructor Once this is placed in, the compiler no longer provides a default constructor causing a compile time error in makeA()

`// to be placed in class A`
`public static void main(String[] args) {`
`    A thingC = new C();`
`    System.out.println(thingC.valC);`
`}`
	compile-time
	Static type of thingC is A and A doesn't have valC.

`// to be placed in class A`
`public static void main(String[] args) {`
`    A thingB = new B();`
`    thingB.banana();`
`}`
	compile-time
	Static type is A and A doesn't have banana()

`// to be placed in class A`
`public static void main(String[] args) {`
`    A thingB = new B();`
`    thingB.g();`
`}`
	ok!
	" h: 2 z: 15

`// to be placed in class B`
`public static void main(String[] args) {`
`    A thingB = new B();`
`    System.out.println(thingB.valA);`
`}`
	Compile time
	ValA is private and class B is not in classA even though B extends A.

// to be placed in class B
public static void main(String[] args) {
    B thingB = new B();
    System.out.println(thingB.h());
}
	ok!
	"2"

// to be placed in class B
public static void main(String[] args) {
    System.out.println(this.valB);
}
	cannot access inst. variable from static context
	– compile time

// to be placed in class B
public static void main(String[] args) {
    System.out.println(B.valB);
}
	valB is an instance variable – compile time

// to be placed in class C
public static void main(String[] args) {
    C thingC = new C();
    System.out.println(thingC.value);
}
	ok! because value is protected.
	"1337"

// to be placed in class C
public static void main(String[] args) {
    C thingC = new A();
    System.out.println(thingC.valC);
}
	compile-time
	something of type A is not necessarily type C.

(Know Your Java) What Type is This?

Look at the code blocks below. In the box besides it, write down the highlighted variables static and dynamic type and then write down what the statement would execute to.

```
public class Main {
    public static void method(B arg1) {
        System.out.println(arg1.g());
    }

    public static void main(String[] args) {
        C thingC = new C();
        method(thingC);
    }
}
```

Static B

dynamic C

```
public class Main {
    public static void method(A arg1) {
        C arg2 = (C) arg1;
        arg2.f();
    }

    public static void main(String[] args) {
        C thingC = new C();
        method(thingC);
    }
}
```

static C

dynamic C

```
public class Main {
    public static void main(String[] args) {
        A thingC = new C();
        thingC.f();
    }
}
```

Static A

dynamic C

```
public class Main {
    public static void main(String[] args) {
        A thing = new B();
        B thingy = (B) thing;
        thingy.banana();
    }
}
```

static B

dynamic B.

(Know Your Java) Functions as Objects

Inspired by Hilfinger 2015 - Unlike Python, Java doesn't treat functions as first class objects. This means that we can't simply pass methods into each other in a functional manner. However, we can employ object orientation to get around this. Consider the following class

```
/** IntFunction is an object representing functions that return ints. */
public class IntFunction {
    // an instance variable that represents an incoming argument.
    int arg;
    public void setArg(int arg) {
        this.arg = arg;
    }
    public int apply() {
        return -1;
    }
}
```

We can consider IntFunction as a base class for all functions that accepts an int and returns an int. The actual functionality of the method we wish to represent would go into the instance method apply. By default, apply returns -1. To produce more complicated behavior (such as methods that accept arguments) we can simply extend IntFunction. Consider SquareFunction, a method that squares an input argument.

```
/** IntFunction is an object representing functions that return ints. */
public class SquareFunction {
    public int apply() {
        return arg * arg;
    }
}
```

And finally, we can now apply it in a functional manner!

```
public static void main(String[] args) {
    int[] inputs = {1, 2, 3, 4, 5};
    SquareFunction sf = new SquareFunction();
    for (int i = 0; i < inputs.length; i += 1) {
        sf.setArg(inputs[i]);
        inputs[i] = sf.apply();
    }
    // now inputs = {1, 4, 9, 16, 25};
}
```

1) Define an object representing the isMultipleOf function. That is your object should be able to take in an integers and N, returning the number if it is a multiple of N else returning -1.

```
public class MultipleFunction extends IntFunction {
    // what do I need as instance variables?
    private int mult;

    // wait there's no return type...
    public MultipleFunction(int mult) {
        this.mult = mult;
    }

    public void setMult(int mult) {
        this.mult = mult;
    }

    // what else do I need?

    public int apply() {
        if (arg % mult == 0) {
            return arg;
        }
        return -1;
    }

}
```

2) Define the map function in this main class. The map function should take in any functional object and apply it to each element of an int list. **NOTE** for easy-mode write map iteratively, for *hard-mode* write map recursively

```
public class Main {

    public static void easyMap(IntList list, IntFunction f) {
        for (; list != null; list = list.tail) {
            f.setArg(list.head);
            list.head = f.apply();
        }
    }

    public static void hardMap(IntList list, IntFunctionf) {
        helper(list, f);
    }


    // helper functions may go here.
    public static void helper(IntList list, IntFunction f){
        if (list != null) {
            f.setArg(list.head);
            list.val = f.apply();
            helper(list.~~tail~~ tail, f);
        }
    }

}
```

(Know Your Java) DBCCCCCCCCCCC

Don't be clueless and read the documentation below! Then write the corresponding code that implements the functionality.

```
public class DBC {

    public static void main(String[] args) {
        /* 2. Declare a variable named "a" of type int. */
        int a                                           ;

        /* 3. Assign the value 3 to a.*/
        a = 3                                           ;

        /* 4. Declare another variable named b of type
         * int, initializing with value 5. */
        int  b=5                                        ;

        /* 5. Declare a variable named c and initialize
         * it with the result of invoking the
         * sum method on a and b. */
        int c = sum(a,b)                                ;

        /* 6. Declare a variable named chaka, which is
         * an array of bytes, with size 7. */
        byte[]  Chaka = new byte[7]                     ;

        /* 7. Declare a variable named khan, which is
         * an array of longs, with contents 1, 2, 3, 4,
         * 5. */
        long[]  khan  = {1,2,3,4,5}                     ;

        /* 8. Set khan's second item (index 1) as chaka's
         * last item. */
        khan[1]  =  chaka[6]                            ;

        /* 9. You are given the following array with unknown
         * size. Write a for-loop that prints all elements
         * IN REVERSE */
        int[] someArray = ...;
        for (int i= someArray.length-1 ; i >=0 ; i+=1) {
            System.out.println(someArray[i]);
        }
        /* 10. You are given an integer n. Using a for
         * loop, create an n by n identity matrix called
         * eye, where all diagonal entries are set to 1
         * and other entries are set to 0 from top left
         * to bottom right. You can represent the matrix
         * as an array of int arrays. Hint: the default
         * value for int is 0. */
        int n = ...;
        int [][] eye = new int[n][n]                   ;
        for (int i=0; i<n; i+=1) {                     ;
           eye[i][i] = 1                               ;
        }                                              ;
        ________________________________________________;
    }
}
```

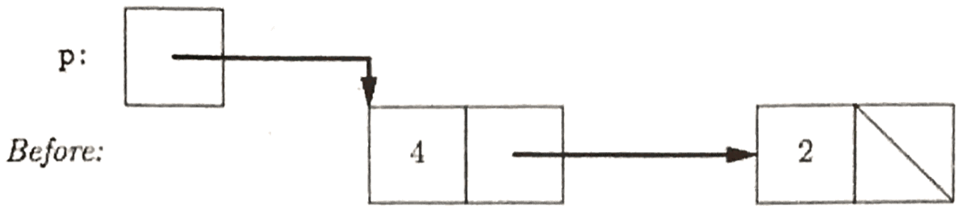
← you don't need these ;

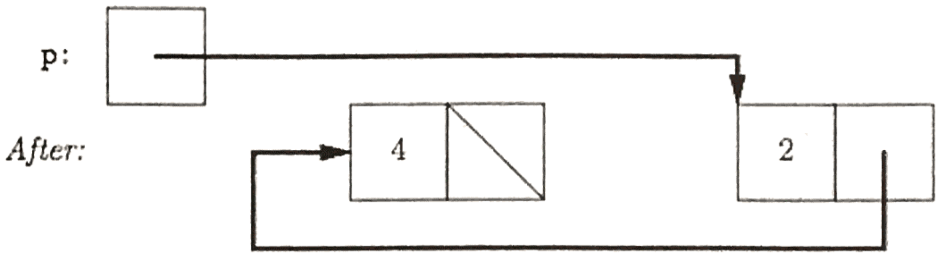
// whoops forgot to include this...

```
static int sum (int a, int b) {
    return a+b;
}
```

(Know Your Java) Pox and Bointers

The before picture shows the state of an IntList object and one local variable. In the lines provided, write the Java code statements that transforms the before picture to the after, subject to the following restrictions. (1) do not modify the _head instance variable and (2) do not introduce any new variables.





```
// code goes here
p.tail.tail = p                          ;
p = p.tail                               ;
p.tail.tail = null                       ;
```

(Array) Easy Mode

These questions test basic understanding of manipulating arrays and such.

Hella Deep Copy Given a 3D array, make a deep copy of the array. That is return a new 3D array that contains the same elements as the previous. Remember the array may be ragged!

```
public static int[][][] hellaDeepCopy(int[][][] array) {
    int[][][] copy = new int[array.length][][];
    for (int i = 0; i < array.length ; i += 1) {
        copy[i] = new int[array[i].length][];
        for ( int j = 0; j < array[i].length; j+=1 ) {
            copy[i][j] = new int[array[i][j].length];
            for (int k = 0; k < array[i][j] .length; k+=1){
                copy[i][j][k] = array[i][j][k];
            }
        }
}
    }
    return copy;
```

Print All Given a 2D array, print all the elements. Remember it's possible that the array is ragged! Make sure to line break each int[] iterated through.

```
public static void printAll(int[][] array) {
    for (int i = 0 ; i < array.length; i+=1) {
        for (int j = 0; j < array[i]. length; j+=1){
            System.out.println( array[i][j] + "  ");
        }
        System.out.println();
    }

}
```

(Array) Medium Mode

This section provides questions that should match your understanding of arrays after labs.

Iterative Fibonacci Array given a number *N* return an array of arrays such that the ith array contains all fibonacci numbers up to the ith one. For example if the N=4, the return result would be [[1], [1, 1], [1, 1, 2], [1, 1, 2, 3]].

```
public static int[][] fibonacciArray(int n) {
    int[][] result = new int[n][];
    for (int i = 0; i<n; i+=1) {
        result[i] = new int[i+1];
        for (int j = 0; j<= i; j+=1) {
            if (j==0 || j==1) {
                result[i][j] = 1;
            } else {
                result[i][j] = result[i][j-1] + result[i][j-2];
            }
        }
    }
    return result;
}
```

IntList to Array convert an IntList to an array.

```
// assume that IntList has a size() method returning the length of the
list
public static int[] toArray(IntList list) {
    int[] result = new int[list.size()];
    toArrayHelper(0, result, list);
    return result;
}

public static void toArrayHelper(int index, int[] nList, IntList list) {
    if (list == null) {
        return;
    } else {
        nList[index] = list.head;
        toArrayHelper(index+1 ,nlist, list.tail);
    }
}
```

(Array) Hard Mode

This last section contains non-trivial, purposefully obfuscated applications of arrays. Try your best!

Recursive Fibonacci Array Do the fibonacci array question again, but recursively!

```
public static int[][] fibonacciArray(int n) {
    int[][] results = new int[n][];
    fibArrayHelper1(results, n-1);
    return results;
}

public static void fibArrayHelper1(int[][] something, int k) {
    if ( k >= 0 ) {
        something[k] = new int[k+1];
        fibArrayHelper2(something[k], k);
        fibArrayHelper1(something, k-1);
    }
}

public static void fibArrayHelper2(int[] array, int k) {
    if ( k == 0 ) {
        array[index] = 1;
    } else if (k == 1) {
        fibArrayHelper2(k - 1, array);
        array[index] = 1;
    } else {
        fibArrayHelper2(array, k-1);
        fibArrayHelper2(array, k-2);
        array[index] = array[k-1] + array[k-2];
    }
}
```

(LinkedList) Easy Mode

The next couple of questions will have some linked list questions. The first few will focus on basic understanding of linked lists.

Deep Copy given an IntList, return an exact copy of that linked list.

```
public static IntList deepCopy(IntList list) {
     if (list == null) {
          return null;
     } else {
          IntList result = new IntList(list.head,
               deepCopy(list.tail));
}         return result;
     }
// in case you want helper methods
```

Find Min Given a IntDList return the max element of that list.

```
public static int maxElement(IntDList list) {
     DNode front = list.front;
     int max = Integer. MIN_VALUE;
     while (front != null) {
          if (front.head > max) {
               max = front.head;
          } front = front.next;
}    }
     return max
```

(LinkedList) Medium Mode

A few questions that should match on par understanding after lab and practice

Copy Odd Given an IntList, non-destructively remove all even elements.

```
public static IntList copyOdd(IntList list) {
    ~~if (list == null) {~~
        ~~return (null);~~
    ~~} else {~~
      ~~IntList result = new IntList ( list.head~~,
          ~~keep~~

}

// in case you want helper methods
Over here !
public static IntList copyOdd (IntList list) {
    if (list == null) {
        return null;
    } else if ( list.head % 2 == 1 ) {
        return new IntList ( list.head,
            copyOdd ( list . tail ) ) ;
    } else {
        return copyOdd ( list . tail ) ;
    }
}
```

Map Write the map function for IntDList and IntList. Map takes in an IntFunction object and IntList/IntDList and applies it to each element of the list.

```
public static void map(IntList list, IntFunction func) {
    if (list != null) {
        func.setArg(list.head);
        list.val = func.apply();
        map(list.tail, func);
    }
}

public static void map(IntDList list, IntFunction func) {
    mapHelper (list.front, func);
}

public static void mapHelper(DNode n, IntFunction f) {
    while (n != null) {
        f.setArg (n.val);
        n.val = f.apply();
        n = n.next;
    }
}
```

" no tears ... only dreams "

– Abraham Lincoln .