CS61B Spring 2016 Guerrilla Section 1 Worksheet SOLUTIONS

Mike Aboody, Leo Colobong, Colby Guan, Laura Harker,
Jimmy Lee, Maurice Lee, Daniel Sochor, Megan Zhu

6 February 2016

Directions: In groups of 4-5, work on the following exercises. Do not proceed to the next exercise until everyone in your group has the answer and *understands why the answer is what it is.* Of course, a topic appearing on this worksheet does not imply that the topic will appear on the midterm, nor does a topic not appearing on this worksheet imply that the topic will not appear on the midterm.

1 Grandpuppies!

Given the following block of code, answer the following questions.

```java
public class Dog {
    public String name;

    public Dog(String name){
        this.name = name;
    }

    public Dog giveBirth(){
        return new Dog(this.name + "'s puppy");
    }

    public void bark(){
        System.out.println(this.name + " barks!");
    }

    public static void main(String args[]) {
        Dog[] myDogs = new Dog[3];
        //Your code inserted here.
    }
}
```

(a) Given the above code, what would you write in the main method to populate `myDogs` with 2 new Dogs named Fido and Fiddle?
myDogs[0] = new Dog(Fido);
myDogs[1] = new Dog(Fiddle);

(b) How would you make Fido's grand-child (the puppy of Fido's puppy) bark, in only one line of code?
myDogs[0].giveBirth().giveBirth().bark();

(c) What would your answer to (b) output?
Fido's puppy's puppy barks!

(d) What would happen if we tried `myDogs[2].bark()`?
NullPointerException

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

2 Bugfixes

Fix the bugs in the Knapsack class below, so that main prints out ”Doge coin:100.45”

Solution:

```java
class Knapsack {
    public String thing;
    public double amount; //Changed

    public Knapsack (String str, double amount) {
        this.thing = str; //Changed
        this.amount = amount; //Changed
    }

    public Knapsack(String str) {
        this(str, 100.45); //Changed
    }

    public static void main (String[] args) {
        Knapsack sack = new Knapsack("Doge coin");
        System.out.println(sack.thing + " : " + sack.amount); //Changed
    }
}
```

STOP!

Don’t proceed until everyone in your group has finished and understands all exercises in this section!

3 Referencing Objects

Draw a box-and-pointer diagram for the execution of `Swap.main`. What is printed when you compile and run this code?

Reminder: Java variables are simple containers that can hold either primitive values (e.g. int, double, char) or references to objects. A reference variable is a 64-bit box that contains the "address" in memory of an instance of a class. 64-bit addresses are meaningless to humans, so we'll represent them with arrows.
All method and constructor calls are pass-by-value, which means that Java copies bits from the caller's variable containers to the callee's argument variable containers (regardless of whether these variables represent primitive data types or references).

```java
public class Point {
    public int x;
    public int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

public class Swap {
    static void swapPoint(Point p) {
        int temp = p.x;
        p.x = p.y;
        p.y = temp;
    }

    static void swap(int a, int b) {
        int temp = a;
        a = b;
        b = temp;
    }

    public static void main(String args[]) {
        int x = 30;
        int y = 19;
        System.out.println("Original x: " + x);
        System.out.println("Original y: " + y);
        swap(x, y);
        System.out.println("New x: " + x);
        System.out.println("New y: " + y);
        Point p = new Point(x, y);
        System.out.println("Original x: " + p.x);
        System.out.println("Original y: " + p.y);
        swapPoint(p);
        System.out.println("New x: " + p.x);
        System.out.println("New y: " + p.y);

    }

}
```

```
Original x: 30
Original y: 19
New x: 30
New y: 19
```

```
Original x: 30
Original y: 19
New x: 19
New y: 30
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

4 Cat World Domination

Toby wants to rule the world with cats. To do this, he's built a `Cat` class to start building his army. In his army of cats, there is a family hierarchy where each cat may or may not have only one parent, and may or may not have kitties (stored in the form of an `ArrayList`). Each cat that has a parent is also a kitty of that parent.

To speed up the world domination process, Toby builds a method called `copyCat` that, in addition to copying the cat, also copies of that cat's descendants. Toby tries to copy his cats initially, but realizes it doesn't work the way he expects it to. Here is his `Cat` class:

```java
public class Cat {
    private Cat parent;
    private ArrayList<Cat> kitties;
    private String name;

    public Cat(Cat parent, String name) {
        this.name = name;
        this.kitties = new ArrayList<Cat>() ;
        this.parent = parent;
    }

    public Cat copyCat() {
        Cat copy = new Cat(this.parent, this.name);
        for (int i = 0; i < this.kitties.size(); i += 1) {
            copy.kitties.add(this.kitties.get(i).copyCat());
        }
        return copy;
    }
}
```

What's wrong with his Cat class? Drawing a box and pointer diagram may help!

While the parent to child relationships are all correct, the copied child to parent relationships are not. In other words, all of the copied `kitties Arraylist<Cat>`s are populated correctly, but the `Cat parent` is not — it is never reassigned, and thus still points to the old parent.

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

5 Arrays

```java
class Foo {
    int x;
    int y;
}
public class ArraysQuestion {
    public static void main(String[] args) {
        int N = 3;
        Foo[] xx = new Foo[N];
        Foo[] yy = new Foo[N];
        for (int i = 0; i < N; i++) {
            Foo f = new Foo();
            f.x = i; f.y = i;
            xx[i] = f;
            yy[i] = f;
        }
        for (int i = 0; i < N; i++) {
            xx[i].y *= 2;
            yy[i].x *= 3;
        }
    }
}
```

After executing the above block of code, what are the values of each `Foo` in `xx` and `yy`?

`xx[0]`: 0 0 `yy[0]`: 0 0

`xx[1]`: 3 2 `yy[1]`: 3 2

`xx[2]`: 6 4 `yy[2]`: 6 4

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

6 Triangularize

Write triangularize, a method that takes in an array of IntLists R and a single IntList L, and breaks L into smaller IntLists, storing them into R. The IntList at index k of R has at most k + 1 elements of L, in order. Thus concatenating all of the IntLists in R together in order would give us L back. Assume R is big enough to do this. For example, if the original L contains [1, 2, 3, 4, 5, 6, 7], and R has 6 elements, then on return R contains [[1], [2,3], [4,5,6], [7], [], []]. If R had only 2 elements, then on return it would contain [[1], [2,3]]. triangularize may destroy the original contents of the IntList objects in L, but does not create any new IntList objects. Note: Assume R's items are all initially null.

Solution:

```java
public static void triangularize(IntList[] R, IntList L) {
    // One of many possible solutions
    int i, k; // i: index into R, k: number of items in R[k]
    i = 0; k = 0;
    while (i < R.length) {
        if (k == 0) {
            R[i] = L;
        }
        if (L == null) {
            i += 1;
            k = 0;
        }
        else if ( k == i) {
          IntList next = L.tail;
          L.tail = null;
          L = next;
          i += 1;
          k = 0;
        }
        else {
            L = L.tail;
            k += 1;
        }
    }
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

7 Mystery

What does the `mystery` function do? Hint: Draw box and pointers.

```java
public class IntList {
    public int head;
    public IntList tail;

    public IntList(int head0, IntList tail0) {
        head = head0; tail = tail0;
    }

    public static IntList mystery(IntList L) {
        if (L == null || L.tail == null) {
            return L;
        } else {
            IntList x = mystery(L.tail);
            L.tail.tail = L;
            L.tail = null;
            return x;
        }
    }

    public String toString() {
        String result = "";
        IntList y = this;
        while (y != null) {
            result = result + y.head + " ";
            y = y.tail;
        }
        return result;
    }

    public static void main(String[] args) {
        IntList x = new IntList(2, new IntList(3, new IntList(4, new IntList(5,
            null))));
        System.out.println(x);
        IntList y = mystery(x);
        System.out.println(y);
    }
}
```

It destructively reverses L, and returns the new head. In this case, running main would cause this output:
2 3 4 5
5 4 3 2

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

8 Braid

Write `braid`, a method that takes in two `IntDLists` of equal length and interleaves the linked lists such that they're a circularly list, then returns the new start node. For example, given `a = [1, 2, 3]` and `b = [4, 5, 6]`, it should return `[1, 5, 3, 4, 2, 6]`.

Solution:

```java
public static IntDList braid(IntDList a, IntDList b) {
    if (a == null) {
        return null;
    }
    IntDList aStart = a;
    IntDList bStart = b;
    IntDList tempNext;
    int count = 0;
    while (a.next != null) {
        tempNext = a.next;
        a.next = b.next;
        a.next.prev = a;
        a = a.next;
        b.next = tempNext;
        b.next.prev = b;
        b = b.next;
        count++;
    }
    a.next = bStart;
    b.next = aStart;
    bStart.prev = a;
    aStart.prev = b;


    return aStart;
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

9 Sorting Zeros and Ones (bonus)

Write `sortZerosOnes`, a method that takes an `Intlist` of only 0s and 1s and sorts the nodes of the list, and returns the new start node. Do not change any head values or create any new nodes. You can assume that the list `L` passed in has only 0s or 1s.

Solution:

```java
public static IntList sortZerosOnes(IntList a) {
    Intlist firstZero = null, lastZero = null, one = null, tail;
    while(a != null) {
        tail = a.tail;
        if(a.head == 0) {
            a.tail = firstZero;
            firstZero = a
            if(lastZero == null) {
                lastZero = a;
            }
        }
        else {
            a.tail = one;
            one = a;
        }
        a = tail;
    }
    if(lastZero != null) { // handle case where there are no zeros
        lastZero.tail = one;
        return firstZero;
    }
    return one;
}
```

STOP!

Don't proceed until everyone in your group has finished and understands all exercises in this section!

10 2D Arrays (bonus)

(a) Write `diagonalFlip`, a method that takes a 2D array `arr` of size NxN and DESTRUCTIVELY flips `arr` along the diagonal line from the left bottom to right top.

Solution:

```
public static void diagonalFlip(int[][] arr){
    int N = arr.length;
    for (int i = N - 1; i >= 0; i--) {
        for (int j = 0; j < N - i - 1; j++) {
            int temp = arr[i][j];
            arr[i][j] = arr[N - j - 1][N - i - 1];
            arr[N - j - 1][N - i - 1] = temp;
        }
    }
}
```

(b) Write `rotate`, a method that takes a 2D array `arr` of size NxN and DESTRUCTIVELY rotates `arr` 90 degrees clockwise.

Solution:

```
public static void rotate(int[][] arr) {
    int N = arr.length;
    for (int i = 0; i < N/2; i++) {
        for (int j = i; i < N - j - 1; j++) {
            int temp = arr[i][j];
            arr[i][j] = arr[N-(j+1)][i];
            arr[N-(j+1)][i] = arr[N-i-1][N-(j+1)];
            arr[N-i-1][N-(j+1)] = arr[j][N-i-1];
            arr[j][N-i-1] = temp;
        }
    }

}
```