

# CS61B SPRING 2016 GUERRILLA SECTION 1 WORKSHEET

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.

```
1 public class Dog {
2     public String name;
3
4     public Dog(String name){
5         this.name = name;
6     }
7
8     public Dog giveBirth(){
9         return new Dog(this.name + "'s puppy");
10    }
11
12    public void bark(){
13        System.out.println(this.name + " barks!");
14    }
15
16    public static void main(String args[]) {
17        Dog[] myDogs = new Dog[3];
18        //Your code inserted here.
19    }
20 }
```

- (a) Given the above code, what would you write in the main method to populate `myDogs` with 2 new Dogs named Fido and Fiddle?

(b) How would you make Fido's grand-child (the puppy of Fido's puppy) bark, in only one line of code?

(c) What would your answer to (b) output?

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

**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"

```
1 class Knapsack {
2     public String thing;
3     public String amount;
4
5     public Knapsack(String str, double amount) {
6         String thing = str;
7         amount = amount;
8     }
9
10    public Knapsack(String str) {
11        Knapsack(str, 100.45);
12    }
13
14    public static void main(String[] args){
15        Knapsack sack = new Knapsack("Doge coin");
16        System.out.println(thing + " : " + amount);
17    }
18 }
```

**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).

```
1 public class Point {
2     public int x;
3     public int y;
4
5     public Point(int x, int y) {
6         this.x = x;
7         this.y = y;
8     }
9 }
10
11 public class Swap {
12     static void swapPoint(Point p) {
13         int temp = p.x;
14         p.x = p.y;
15         p.y = temp;
16     }
17
18     static void swap(int a, int b) {
19         int temp = a;
20         a = b;
21         b = temp;
22     }
23
24     public static void main(String args[]) {
25         int x = 30;
26         int y = 19;
27         System.out.println("Original x: " + x);
28         System.out.println("Original y: " + y);
29         swap(x, y);
30         System.out.println("New x: " + x);
31         System.out.println("New y: " + y);
32         Point p = new Point(x, y);
33         System.out.println("Original x: " + p.x);
34         System.out.println("Original y: " + p.y);
35         swapPoint(p);
36         System.out.println("New x: " + p.x);
37         System.out.println("New y: " + p.y);
38     }
39 }
40
41 }
```

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:

```
1 public class Cat {
2     private Cat parent;
3     private ArrayList<Cat> kitties;
4     private String name;
5
6     public Cat(Cat parent, String name) {
7         this.name = name;
8         this.kitties = new ArrayList<Cat>();
9         this.parent = parent;
10    }
11
12    public Cat copyCat() {
13        Cat copy = new Cat(this.parent, this.name);
14        for (int i = 0; i < this.kitties.size(); i += 1) {
15            copy.kitties.add(this.kitties.get(i).copyCat());
16        }
17        return copy;
18    }
19 }
```

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

**STOP!**

DON'T PROCEED UNTIL EVERYONE IN YOUR GROUP HAS FINISHED AND UNDERSTANDS ALL EXERCISES IN THIS SECTION!

## 5 Arrays

```
1 class Foo {
2     int x;
3     int y;
4 }
5 public class ArraysQuestion {
6     public static void main(String[] args) {
7         int N = 3;
8         Foo[] xx = new Foo[N];
9         Foo[] yy = new Foo[N];
10        for (int i = 0; i < N; i++) {
11            Foo f = new Foo();
12            f.x = i; f.y = i;
13            xx[i] = f;
14            yy[i] = f;
15        }
16        for (int i = 0; i < N; i++) {
17            xx[i].y *= 2;
18            yy[i].x *= 3;
19        }
20    }
21 }
```

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

xx[0]:                      yy[0]:

xx[1]:                      yy[1]:

xx[2]:                      yy[2]:

**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`.

```
1 public static void triangularize(IntList[] R, IntList L) {
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25 }
```

**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.

```
1 public class IntList {
2     public int head;
3     public IntList tail;
4
5     public IntList(int head0, IntList tail0) {
6         head = head0; tail = tail0;
7     }
8
9     public static IntList mystery(IntList L) {
10        if (L == null || L.tail == null) {
11            return L;
12        } else {
13            IntList x = mystery(L.tail);
14            L.tail.tail = L;
15            L.tail = null;
16            return x;
17        }
18    }
19
20    public String toString() {
21        String result = "";
22        IntList y = this;
23        while (y != null) {
24            result = result + y.head + " ";
25            y = y.tail;
26        }
27        return result;
28    }
29
30    public static void main(String[] args) {
31        IntList x = new IntList(2, new IntList(3, new IntList(4, new IntList(5,
32            null)));
33        System.out.println(x);
34        IntList y = mystery(x);
35        System.out.println(y);
36    }
```

**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]`, the return should be `[1, 5, 3, 4, 2, 6]`.

```
1 public class IntDList {
2     public int val;
3     public IntDList prev, next;
4 }
5
6 public static IntDList braid(IntDList a, IntDList b) {
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30 }
```

**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.

```
1 public static IntList sortZerosOnes(IntList a) {  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25 }
```

**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  $N \times N$  and DESTRUCTIVELY flips `arr` along the diagonal line from the left bottom to right top.

```
1 public static void diagonalFlip(int [][] arr){
2
3
4
5
6
7
8
9
10
11
12
13
14 }
15
```

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

```
1 public static void rotate(int [][] arr) {
2
3
4
5
6
7
8
9
10
11
12
13
14 }
15
```