Instructions

Form a small group. Start on the first problem. Check off with a helper or discuss your solution process with another group once everyone understands how to solve the first problem and then repeat for the second problem ...

You may not move to the next problem until you check off or discuss with another group and everyone understands why the solution is what it is. You may use any course resources at your disposal: the purpose of this review session is to have everyone learning together as a group.

1. Take a Knap, hit the sack

1.1 Fix the bugs in Knapsack so that main prints out Doge coin : 100.45.

```java
class Knapsack {
    public String thing;
    public String amount;

    public Knapsack(String str, double amount) {
        thing = str;
        amount = amount;
    }

    public Knapsack(String str) {
        Knapsack(str, 100.45);
    }

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

Toby wants to rule the world with an army of cats. Each cat may or may not have one parent, and may or may not have ‘kitties’. Each cat that has a parent is a ‘kitty’ of that parent. But after implementing `copyCat`, which creates a copy of a cat and its descendants, he realizes the function contains a bug.

```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!
3 Some Sort of Interface

3.1 Suppose we’d like to implement the SortedList interface.

```java
public interface SortedList {
    /* Return the element at index i, the ith (0-indexed) smallest element. */
    int get(int i);

    /* Remove the element at index i, the ith (0-indexed) smallest element. */
    int remove(int i);

    /* Insert an element into the SortedList, maintaining sortedness. */
    void insert(int elem);

    /* Return the size of the SortedList. */
    int size();
}
```

(a) Suppose we’d like to optimize the speed of `SortedList::get`. Should we implement `SortedList` with a linked list or an internal array?

(b) Implement the default method, `merge`, which takes another `SortedList` and merges the other values into the current `SortedList`.

```java
default void merge(SortedList other) {
```

(c) Suppose we’d like to `merge` using only a constant amount of additional memory. Should we implement `SortedList` with a linked list or an internal array?

(d) Implement the default method, `negate`, which destructively negates all the values in the `SortedList`.

```java
default void negate() {
```
4 Arrayana Grande

4.1 After executing the code, what are the values of \texttt{Foo} in \texttt{xx} and \texttt{yy}?

```java
public class Foo {
    public int x, y;

    public static void main(String[] args) {
        int N = 3;
        Foo[] xx = new Foo[N], yy = new Foo[N];
        for (int i = 0; i < N; i++) {
            Foo f = new Foo();
            f.x = 1; f.y = i;
            xx[i] = f;
            yy[i] = f;
        }
        for (int i = 0; i < N; i++) {
            xx[i].y *= 2;
            yy[i].x *= 3;
        }
    }
}
```

(a) \texttt{xx[0]}  (d) \texttt{yy[0]}
(b) \texttt{xx[1]}  (e) \texttt{yy[1]}
(c) \texttt{xx[2]}  (f) \texttt{yy[2]}

5 Just another friendy...

5.1 What is the output after running the \texttt{main} method in the \texttt{Ghoul} class?

```java
public class Ghoul {
    public String noise = "blargh";
    public static int spookFactor = 5;

    public Ghoul() {
        System.out.println("Muhahaha!!!");
    }

    public void spook(Monster m) {
        System.out.println("I go " + noise);
    }

    public void spook(Ghoul g) {
        System.out.println("I am " + spookFactor + " spooky.");
    }
}
```
public class Ghoul extends Monster {
    public Ghoul() {
        System.out.println("I am a ghoul");
    }

    public void spook(Ghoul g) {
        System.out.println("I'm so ghoul");
        ((Monster) g).spook(g);
    }

    public void haunt() {
        Monster m = this;
        System.out.println(noise);
        m.spook(this);
    }

    public static void main(String[] args) {
        Monster m = new Monster();
        m.spook(m);

        Monster g = new Ghoul();
        g.spook(m);
        g.spook(g);

        Monster.spookFactor = 10;
        Ghoul ghastly = new Ghoul();
        ghastly.haunt();
    }
}
public interface BinaryFunction {
    public int apply(int x, int y);
}

public interface UnaryFunction {
    public int apply(int x);
}

6.1 Implement Adder, which implements the BinaryFunction interface and adds two numbers together.

   public class Adder

6.2 Implement Add10 which implements UnaryFunction. Its apply method returns x + 10 without using any of the + - * / operators.

   public class Add10

6.3 Implement Multiplier which implements BinaryFunction. Its apply method accepts two integers, x and y, and return x * y without using any of the + - * / operators except to increment indices in loops. Assume all inputs are positive.

   public class Multiplier
7 Tri another angle *Extra*

Implement `triangularize`, which takes an `IntList[] 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`.

```java
public static void triangularize(IntList[] R, IntList L) {
```
8 Arrays of the 2D variety Extra

8.1 Implement diagonalFlip, a method that takes a 2-D array arr of size $N \times N$ and destructively flips arr along the diagonal line from the left bottom to right top.

```java
public static void diagonalFlip(int[][] arr) {
```

8.2 Implement rotate, which takes a 2-D array arr of size $N \times N$ and destructively rotates arr 90 degrees clockwise.

```java
public static void rotate(int[][] arr) {
```