CS 61B Spring 2018

Midterm 1 Guerrilla Section

February 10, 2018

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`.

```
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 }
```

2 I Like Cats

2.1 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.

```
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!

3 Some Sort of Interface

3.1 Suppose we'd like to implement the `SortedList` interface.

```
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`.

```
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`.

```
default void negate() {
```

4 Arrayana Grande

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

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

(a) `xx[0]`
(b) `xx[1]`
(c) `xx[2]`
(d) `yy[0]`
(e) `yy[1]`
(f) `yy[2]`

5 Just another fiendly...

5.1 What is the output after running the `main` method in the `Ghoul` class?

```
1  public class Monster {
2      public String noise = "blargh";
3      public static int spookFactor = 5;
4
5      public Monster() {
6          System.out.println("Muhahaha!!!");
7      }
8
9      public void spook(Monster m) {
10         System.out.println("I go " + noise);
11     }
12
13     public void spook(Ghoul g) {
14         System.out.println("I am " + spookFactor + " spooky.");
15     }
16 }
```

```java
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();
    }
}
```

6 David HasselHoF

```
1 public interface BinaryFunction {
2     public int apply(int x, int y);
3 }
```

```
1 public interface UnaryFunction {
2     public int apply(int x);
3 }
```

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

```
1 public class Adder
```

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

```
1 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.

```
1 public class Multiplier
```

7 Tri another angle *Extra*

7.1 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`**.

```
1 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.

```
1 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.

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