CS 61B Spring 2017

Generics & Exceptions

Discussion 6: February 22, 2017

1 AltList

1.1 A normal generic linked list contains objects of only one type. But we can imagine a generic linked list where entries alternate between two types.

```
public class AltList<X,Y> {
    private X item;
    private AltList<Y,X> next;
    AltList(X item, AltList<Y,X> next) {
        this.item = item;
        this.next = next;
    }
}

AltList<Integer, String> list =
    new AltList<Integer, String>(5,
        new AltList<String, Integer>("cat",
            new AltList<Integer, String>(10,
                new AltList<String, Integer>("dog", null))));
```

This list represents `[5, cat, 10, dog]`. In this list, assuming indexing begins at 0, all even-index items are `Integers` and all odd-index items are `Strings`.

Write an instance method called `pairsSwapped()` for the `AltList` class that returns a copy of the original list, but with adjacent pairs swapped. Each item should only be swapped once. This method should be non-destructive: it should not modify the original `AltList` instance. Assume that the list has an even, non-zero length.

For example, calling `pairsSwapped()` on the list `[5, cat, 10, dog]` should yield the list `[cat, 5, dog, 10]`.

```
public class AltList<X,Y> {
    public AltList<Y,X> pairsSwapped() {
        AltList<Y,X> ret = new AltList<Y,X>(next.item, new AltList<X,Y>(item, null));
        if (next.next != null) {
            ret.next.next = next.next.pairsSwapped();
        }
        return ret;
    }
}
```

2 Mercantilism

2.1 Let's model a feudal society where managers, merchants, and workers cooperate to deliver goods. What happens when we call `new Manager().work()`?

```
class Manager {
    Merchant merchant = new Merchant();
    Worker worker = new Worker();
    String[] goods = new String[1];
    void work() {
        try {
            merchant.trade(goods);
        } catch (RuntimeException e) {
            worker.produce("apple pie", goods);
            merchant.trade(goods);
            worker.produce("cornbread", goods);
        } catch (Exception e) {
            merchant.trade(goods);
        } finally {
            System.out.println("All in a day's work");
        }
    }
}
class Merchant {
    void trade(String[] goods) {
        try {
            for (String good : goods) {
                if (good != null) {
                    System.out.println("Traded 1 " + good);
                    return;
                }
            }
            throw new RuntimeException("Not enough goods");
        } catch (Exception e) {
            System.out.println("Oops");
            throw e;
        } finally {
            System.out.println("I love trading");
        }
    }
}
class Worker {
    void produce(String item, String[] goods) {
        int i = 0;
        while (goods[i] != null) {
            i += 1;
        }
        goods[i] = item;
        System.out.println("Done making 1 " + item);
    }
}
```

```
Oops
I love trading
Done making 1 apple pie
Traded 1 apple pie
I love trading
All in a day's work
ArrayIndexOutOfBoundsException
```