Homework 9

Due: Monday, December 1st, 6pm

Task 1 – We Game, We Saw, We Conquered [8 pts]

Use the classes from Task 1 of section to answer these questions. State whether each statement is true or false. Recall that A is a Java subtype of B if Java will let you pass an A where a B is expected. This is implicit casting an A into a B. If an explicit cast is required, it is not a subtype.

a) `Set<CardGame>` is a Java subtype of `Collection<CardGame>`

b) `Collection<Game>` is a Java subtype of `Set<Game>`

c) `Set<VideoGame>` is a Java subtype of `Set<Game>`

d) `Set<Game>` is a Java subtype of `Set<VideoGame>`

e) `Set<VideoGame>` is a Java subtype of `Collection<Game>`

f) `Set<VideoGame>` is a Java subtype of `Collection<VideoGame>`

g) `VideoGame[]` is a Java subtype of `Game[]`

h) `Game[]` is a Java subtype of `VideoGame[]`

Task 2 – They've Got a Max To Grind [8 pts]

In this problem, we will generalize `MaxStack` (see Task 3 of section) further into a `ReduceStack`, replacing the max operation with an arbitrary binary operation $r : (T, T) \to T$. We will use that operation to "reduce" a list of values to a certain value by applying r multiple times. For example, the list $1 :: 2 :: 3 :: 4 :: \text{nil}$ would be reduced to $r(1, r(2, r(3, 4)))$.

The client using `ReduceStack` will need to give us this operation somehow. The way to do that in Java is via an interface. It could look like this, for example:

```
public interface Reducer<T> {
    T reduce(T a, T b);
}
```

If `myReducer` is an instance of this interface, then we could call `myReducer.reduce(a, b)` to calculate $r(a, b)$ above.

a) In order to implement this class in the same manner as we did for `MaxStack`, where each `push` and `pop` take constant time, how should the client give us the `Reducer`?

b) Fill in the specification for the `reduceAll` method in the `ReduceStack` interface below, assuming that r is the known binary operation used to reduce. Recall that the abstract state of the stack can be thought of as a list of elements of the established type. As a hint, your return should define a mathematical function reduceAll : List<T> $\to T$, which is different from the code method, and utilize the given binary operator r.

```
public interface ReduceStack<T> {
    ...

    /* Your specification of reduce goes here */
    T reduceAll();

    ...
}
```

Task 3 – The Whole Ball of Max [17 pts]

In this problem, we will finish our work by updating `MaxStackImpl` into `ReduceStackImpl`.

As before, `PairList` is declared `static`. If that is left out, then each instance includes an extra (hidden) reference to the instance of `ReduceStackImpl` that created it. That is unnecessary, so it would waste memory.

One advantage of not declaring it `static` would be that the `<T>` declared on `ReduceStackImpl` would be in scope here, so we would not need to include a `<T>` on each `PairList`. However, again, that would waste memory, so it is better to declare it `static` and give `PairList` its own `<T>`.

```java
public class ReduceStackImpl<T> implements ReduceStack<T> {
    private final Reducer<T> reducer;
    private PairList<T> head;
    private int size;

    private static class PairList<T extends Comparable<T>> {
        public final T val;
        public final T reduced;
        public final PairList<T> next;

        public PairList(T val, T reduced, PairList<T> next) {
            this.val = val;
            this.reduced = reduced;
            this.next = next;
        }
    }

    public ReduceStackImpl(...) {
        // Your code here
    }

    public int size() {
        return size;
    }

    public void push(T val) {
        // Your code here
    }

    public T pop() {
        // Your code here
    }

    public T reduceAll() {
        // Your code here
    }
```

a) Explain why we need the three fields in `ReduceStackImpl`. Write an RI based on these fields.

b) Fill in the constructor of the class so that it initializes the fields properly, including `reducer`.

c) Fill in the `push` method.

d) Fill in the `pop` method.

e) Fill in the `reduceAll` method.

Task 4 – Extra Credit: Not One False Prove [15 pts]

Our definition of ReduceStack applies the operation starting at end of the list, i.e., from the right end.

Prove that, if the operation r satisfies $r(r(a,b),c) = r(a,r(b,c))$, for any a, b, and c, then the result is the same regardless of which end we start at. Specifically, prove that

$$\mathsf{rleft}(x :: L) = \mathsf{rright}(x :: L)$$

holds for all lists L, where we define as the following:

$$\begin{aligned} \mathsf{rright}(x :: \mathsf{nil}) &:= x \\ \mathsf{rright}(x :: y :: L) &:= r(x, \mathsf{rright}(y :: L)) \end{aligned}$$

$$\begin{aligned} \mathsf{rleft}(x :: \mathsf{nil}) &:= x \\ \mathsf{rleft}(x :: y :: L) &:= \mathsf{rleft}(r(x,y) :: L) \end{aligned}$$