1 Exceptions (Spring 2016 MT2 Q3)

Consider the code below. Recall that \( x / 2 \) rounds down to the nearest integer.

```java
public static void checkIfZero(int x) throws Exception {
    if (x == 0) {
        throw new Exception("x was zero!"),
    } System.out.println(x); // PRINT STATEMENT
}

public static int mystery(int x) {
    int counter = 0;
    try {
        while (true) {
            x = x / 2;
            checkIfZero(x);
            counter += 1;
        System.out.println("counter is " + counter); // PRINT STATEMENT
    }
    catch (Exception e) {
    return counter;
    }
}

public static void main(String[] args) {
    System.out.println("mystery of 1 is " + mystery(1));
    System.out.println("mystery of 6 is " + mystery(6));
}
```

What will be the output when main is run? You may not need all lines.

```
mystery of 1 is 0
3
counter is 1
1
counter is 2
mystery of 6 is 2
```
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. **AltList** is an implementation of such a data structure:

```java
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;
    }
}
```

Let’s construct an AltList instance:

```java
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.

For example, calling `list.pairsSwapped()` should yield the list [cat 5 dog 10]. There were two swaps: "cat" and 5 were swapped, then "dog" and 10 were swapped. You may assume that the list on which `pairsSwapped()` is called has an even non-zero length. Your code should maintain this invariant.

```java
class AltList<X, Y> {
    // ... continued from above

    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;
    }
}
```
Every $K$th Element (Fall 2014 MT1 Q5)

Fill in the `next()` method in the following class. Do not modify anything outside of `next`.

```java
import java.util.Iterator;
import java.util.NoSuchElementException;
/** Iterates over every Kth element of the IntList given to the constructor. For example, if L is an IntList containing elements [0, 1, 2, 3, 4, 5, 6, 7] with K = 2, then for (Iterator<Integer> p = new KthIntList(L, 2); p.hasNext(); ) {
    System.out.println(p.next());
} would print get 0, 2, 4, 6. */
public class KthIntList implements Iterator<Integer> {
    public int k;
    private IntList curList;
    private boolean hasNext;

    public KthIntList(IntList I, int k) {
        this.k = k;
        this.curList = I;
        this.hasNext = true;
    }

    /** Returns true iff there is a next Kth element. Do not modify. */
    public boolean hasNext() {
        return this.hasNext;
    }

    /** Returns the next Kth element of the IntList given in the constructor. Returns the 0th element first. Throws a NoSuchElementException if there are no Integers available to return. */
    public Integer next() {
        if (curList == null) {
            throw new NoSuchElementException();
        }
        Integer toReturn = curList.head;
        for (int i = 0; i < k && curList != null; i++) {
            curList = curList.tail;
        }
        hasNext = (curList != null);
        return toReturn;
    }
}
```