Skippify (Spring '17, MT1)

Here is a video walkthrough of the solutions.

We have the following `IntList` class, as defined in lecture and lab, with an added `skippify` function. Suppose that we define two IntLists as follows.

```
IntList A = IntList.list(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
IntList B = IntList.list(9, 8, 7, 6, 5, 4, 3, 2, 1);
```

Fill in the method `skippify` such that the result of calling `skippify` on A and B are as below:
- After calling `A.skippify()`, A: (1, 3, 6, 10)
- After calling `B.skippify()`, B: (9, 7, 4)

```
public class IntList {
    public int first;
    public IntList rest;

    @Override
    public boolean equals(Object o) { ... }
    public static IntList list(int... args) { ... }

    public void skippify() {
        IntList p = this;
        int n = 1;
        while (p != null) {

            IntList next = ___________________________;

            for (_______________________________________) {

                if (________________________________________) {

                    ___________________________________________
                }

                ___________________________________________
            }

            ___________________________________________

            ___________________________________________

            ___________________________________________
        }
    }
}
```

Solution:

```java
public class IntList {
    public int first;
    public IntList rest;

    @Override
    public boolean equals(Object o) { ... }
    public static IntList list(int... args) { ... }

    public void skippify() {
        IntList p = this;
        int n = 1;
        while (p != null) {
            IntList next = p.rest;
            for (int i = 0; i < n; i += 1) {
                if (next == null) {
                    break;
                }
                next = next.rest;
            }
            p.rest = next;
            p = p.rest;
            n++;
        }
    }
    ...
}
```

Explanation: Looking at `IntList A`, we only need to change the `rest` attribute of `IntList` instances 1, 3, and 6. To achieve this, we will use the **`for`** loop to find the new `rest` attribute (which we will store in `next`) of the current `IntList` instance (`p`). The outer **`while`** loop enables us to repeat these actions for, in our case, `IntList` instances 3 and 6. The **`int`** `n` will increment by one each iteration and gives us the number of iterations in the for loop, i.e. how many `IntList` instances to skip. Finally, the **`if`** check accounts allows us to exit the for loop early if we ever hit the end of the Linked List.