Important

There are general homework guidelines you must always follow. If you fail to follow any of the following guidelines you risk receiving a **0** for the entire assignment.

1. All submitted code must compile under **JDK 8**. This includes unused code, so don't submit extra files that don't compile. Any compile errors will result in a 0.
2. Do not include any package declarations in your classes.
3. Do not change any existing class headers, constructors, or method headers.
4. Do not add additional public methods.
5. Do not use anything that would trivialize the assignment. (e.g. don't import/use `java.util.LinkedList` for a Linked List assignment. Ask if you are unsure.)
6. Always be very conscious of efficiency. Even if your method is to be $O(n)$, traversing the structure multiple times is considered non-efficient unless that is absolutely required (and that case is extremely rare).
7. You must submit your source code, the `.java` files, not the compiled `.class` files.
8. After you submit your files redownload them and run them to make sure they are what you intended to submit. You are responsible if you submit the wrong files.

Skip Lists

You are to code a skip list. The bottom layer of the skip list is an ordinary linked list with every element. The very first level in your skip list is level 1. See the interface for more information regarding implementation.

Duplicate items will not be passed in into `put()`.

Phantom Nodes

You are to implement your skip list with a phantom node at the beginning of each level. The method `getHead()` should return the phantom node on the highest level which will act as the head. Do not have any phantom nodes at the end of each level.

Levels

You must have an empty level at the top of your skip list at all times. If the skip list is empty, then there should still be an empty level; `head` should then point to a phantom node that is not connected to any other node.

Coin Flipper

This will be the source of randomness for your skip list. You MUST use the `CoinFlipper` passed into the given constructor and no other source of randomness (e.g. a `CoinFlipper` you instantiate, a new `Random` object, etc) or you will lose a lot of points. When adding an item to your skip list, it will always be on the first level. After that, you will use the coin flipper to determine if it will be promoted to the next level. Heads means that you promote the item to the next level.

Example: You are adding the string "bananaaaa". You flip `HHT`. That means it will be put on the bottom level and then promoted two levels. It will be on the bottom 3 levels.

String representation of Skip List

The `toString()` method has been provided to help with debugging. This is only meant to help give you a glimpse of what is in your skip list. It is not meant to replace testing.

A note on JUnits

We have provided a **very basic** set of tests for your code, in `SkipListStudentTests.java`. These tests do not guarantee the correctness of your code (by any measure), nor does it guarantee you any grade. You may additionally post your own set of tests for others to use on the Georgia Tech GitHub as a gist. Do **NOT** post your tests on the public GitHub. There will be a link to the Georgia Tech GitHub as well as a list of JUnits other students have posted on the class Piazza.

If you need help on running JUnits, there is a guide, available on T-Square under Resources, to help you run JUnits on the command line or in IntelliJ.

Style and Formatting

It is important that your code is not only functional but is also written clearly and with good style. We will be checking your code against a style checker that we are providing. It is located in T-Square, under Resources, along with instructions on how to use it. We will take off a point for every style error that occurs. If you feel like what you wrote is in accordance with good style but still sets off the style checker please email Jonathan Jemson (jonathanjemson@gatech.edu) with the subject header of "CheckStyle XML".

Javadocs

Javadoc any helper methods you create in a style similar to the existing Javadocs. If a method is overridden or implemented from a superclass or an interface, you may use `@Override` instead of writing Javadocs.

Exceptions

When throwing exceptions, you must include a message by passing in a String as a parameter. **The message must be useful and tell the user what went wrong**. "Error", "BAD THING HAPPENED", and "fail" are not good messages. The name of the exception itself is not a good message.

For example:

```
throw new PDFReadException("Did not read PDF, will lose points.");

throw new IllegalArgumentException("Cannot insert null data into data structure.");
```

Generics

If available, use the generic type of the class; do **not** use the raw type of the class. For example, use `new LinkedList<Integer>()` instead of `new LinkedList()`. Using the raw type of the class will result in a penalty.

Forbidden Statements

You may not use these in your code at any time in CS 1332.

- `break` may only be used in switch-case statements

- `continue`
- `package`
- `System.arraycopy()`
- `clone()`
- `assert()`
- `Arrays` class
- `Array` class
- `Objects` class
- `Collections` class
- `Collection.toArray()`
- Reflection APIs
- Inner, nested, or anonymous classes

Debug print statements are fine, but nothing should be printed when we run them. We expect clean runs - printing to the console when we're grading will result in a penalty. If you use these, we will take off points.

Provided

The following file(s) have been provided to you. There are several, but you will edit only one of them.

1. `SkipListInterface.java`

   This is the interface you will implement in `SkipList`. All instructions for what the methods should do are in the javadocs. **Do not alter this file.**

2. `SkipList.java`

   This is the class in which you will implement `SkipListInterface`. Feel free to add private helper methods but **do not add any new public methods, new classes, instance variables, or static variables**.

3. `CoinFlipper.java`

   This class represents a coin flipper. This class tells you whether or not a data item should be promoted. **Do not alter this file.**

4. `SkipListNode.java`

   This class represents a node in the skip list. It encapsulates the `data`, `level`, `next`, `previous`, `up`, and `down` references. **Do not alter this file.**

5. `SkipListStudentTests.java`

   This is the test class that contains a set of tests covering the basic operations on the `SkipList` class. It is not intended to be exhaustive and does not guarantee any type of grade. **Write your own tests to ensure you cover all edge cases.**

Deliverables

You must submit **all** of the following file(s). Please make sure the filename matches the filename(s) below, and that *only* the following file(s) are present. T-Square does **not** delete files from old uploads; you must do this manually. Failure to do so may result in a penalty.

After submitting, be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder, copy over the interfaces, recompile, and run. It is your responsibility to re-test your submission and discover editing oddities, upload issues, etc.

1. `SkipList.java`