Homework 3

You may **not** use java's built-in queues, stacks or trees.
You also may not use any of java's built-in lists to implement a stack, queue or tree.
Doing either of these is cheating and results in an F in the course. If in doubt, ask on the forum.

The problems are in order of difficulty. The final problem is only worth a few points and intended for A students. For all of these questions, the difficulty is the design/problem solving part, not the program. You may ask the TAs and friends for help with java syntax but not on how to solve the problems.

Permissions

1. You are allowed to add any methods or properties you want.
2. You are not allowed to remove/not implement the ones given in the class diagrams.
3. You may not modify test cases. More precisely, we will always erase your test cases when grading and use our own, so if your code only works on modified tests cases it will not work with ours.
4. If the goal of a question is to implement a data structure, you may not delegate any part of that to a built-in java data structure (this counts as cheating).
5. You may use built-in data structures if it is needed for a public interface (e.g., if you are expected to take in or give out a List, you can import and use List).
6. You may import code from one of your questions to another. This is new for this assignment. Prior to this the grading script didn't allow this. Hopefully we've fixed that.

Rules

1. Submit a zip (not rar, stuffit, etc.) file containing the java source code you wrote (i.e., we don't need your copy of the tests, we already have those).
2. If your code breaks the grading script, you will get a 0 for the assignment. Concretely, do not submit code that doesn't compile or has an infinite loop. Either don't submit those files or make them compile (just return null or -1).

1. **Profiler** (5 points)
Tests: Use of the appropriate data structure
Rank: Easy

In the real world, performance tuning is done using (among other things) a profiler. A profiler reports how much time is spent in each method, helping the tuner determine which code to optimize. We're going to make a poor man's profiler.

`Profiler`
...whatever you need...

`- Profiler() : ctor`
`+ getSingleton() : Profiler`
`+ add(String methodName) : void`
`+ getNumberOfMethods() : int`
`+ getNumberOfMethodCalls() : int`
`+ getNumberOfMethodCalls(String methodName) : int`
`+ getProfile(String methodName) : MethodProfile`
`+ getProfiles() : List<MethodProfile>`

Notes:
- Uses the singleton pattern. You've already done this in HW2 so you should be OK.
- You need to decide which data type to use to store the data. You may not ask which structure it is, you need to figure that out.
- You may use built-in java data structures.

`Profiler()`	The constructor is private. There are no other constructors. The class cannot be externally instantiated.
`getSingleton()`	Lazy loads (gets if it exists, creates if it doesn't) the static, singleton instance variable `instance`.
`add(String methodName)`	Called when a method is entered. If the profiler already has an entry for the method, increment the count, otherwise add it with a count of 1. Text
`getNumberOfMethods()`	How many methods were tracked
`getNumberOfMethodCalls()`	How many times was any method called (i.e., the sum of the counts).
`getNumberOfMethodCalls( String methodName)`	How many times the specified method was called.
`getProfile(String methodName)`	Get the `MethodProfile` of the specified method.
`getProfiles()`	Get the `MethodProfile` for all methods. Extra

	points if they are sorted by from highest (percentage) to lowest. You may sort in any way you can figure out, including built-in methods.

`MethodProfile`

`+ count : int`
`+ name : String`
`+ percentOfCalls: : float`

The profile for a given method is stored in a `MethodProfile`. The profile says how often the method was called and what percentage (0 to 1) of the total number of calls it was.

---

2. Fibonacci Lists (10 points)

Tests: Java, recursion, dictionaries
Rank: Easy

Fibonacci numbers are 0, 1 and numbers that are the sum of the two Fibonacci numbers right before it. The first 12 numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89. You need to write a class that can give you a specified number. For example, `get(12)` returns the 12th Fibonacci number, 89.

FibonacciSequence
+ get(int sequenceID) : int

There are a couple of ways to do this but for this class use the traditional recursive solution. Since i know you'll look at Wikipedia anyway, here's how they describe it:

```
function fib(n)
    if n = 0 return 0
    if n = 1 return 1
    return fib(n - 1) + fib(n - 2)
```

So implement that.

If you think about it, you calculate most numbers multiple times. For example,

```
get(5) = get(4)+get(3)
get(4) = get(3)+get(2)
get(3) = get(2)+get(1)
get(2) = get(1)+get(0)
get(1) = 1
get(0) = 0
```

`get(3)` is calculated twice (by `get(5)` and `get(4)`). `get(2)` is calculated three times:

```
            get(5) = get(4)  +  get(3)
     get(4)=get(3)+get(2)         get(3)=get(2)+get(1)
  get(3)=get(2)+get(1)
```

That's a lot of wasted work and gets worse on larger problems (growth = ~O(2^{n+1}), which is exponential which is icky). We should fix that.

FibonacciSequence
+ get(int sequenceID) : int + getFaster(int sequenceID) : int - getFasterInternal(int sequenceID) : int

How do we go faster? Easy – don't do duplicate work. Before you recursively calculate a number, see if you already know the answer. This implies, of course, that whenever you calculate a number, you should store that answer somewhere where you can get to it later. We'll let you figure out how to do that and which data structure to use. Hint 1: You need to cache the answer at the class level, not in a local variable. Hint 2: You need to store both the sequence ID and the Fibonacci number (the answer) for it.

How do we know that `getFaster()` is faster than `get()`? Time to use that profiler you wrote. In both `get()` and `getFasterInternal()` add the method name to the profiler ("get" for the get(), the slightly inaccurate "getFaster" for getFasterInternal). Don't bother profiling `getFaster()`; it will only be called once per problem and constants are boring.

`get`	Get the specified Fibonacci number using the given recursive approach.
`getFaster`	The method `getFaster()` is split into two parts. This is the first. To accurately measure performance, you need to start with a blank memory. Clear the cache in this method. You could also clear the profiler here but you don't have to, we'll do that in our tests. Once you've prepared the way, call (delegate the work to) `getFasterInternal().`
`getFasterInternal`	This is the second part of `getFaster().` Do the actual recursive calculation here.

Note: Don't just try to pass the unit tests, actually run it. Passing unit tests is the bare minimum you should be doing and the bare minimum is for slackers who end up with second-rate jobs that don't pay nearly as well as the jobs that people who like this stuff get. Write an application to see what the effects of caching information is. The technique is called dynamic programming and essentially converts an exponential-time algorithm into linear-time. It's important, so spend a little time with it.

3. Text Editing Undo (20 points)

Tests: Stacks
Rank: Medium

We're going to make a really simple text editor. Almost all of it you can do with String's build-in methods. What you'll need to implement is the unlimited undo feature.

TextEditor
`+ history : MyStack<Action>`

`+ delete(int first, int last) : void`
`+ getValue() : String`
`+ insert(String value, int position) : void`
`+ replace(int first, int last, String value) : void`
`+ setValue(String value) : void`
`+ hackerSetValue(String value) : void`
`+ undo() : void`

When first created, the value of `TextEditor` is the empty string "". The text in the text editor is 1-based, not 0-based, so character 1 in "ABC" is A.

`delete`	Delete the characters between the specified indices, inclusive. Example: delete(2,4) in the string "12345" removes 2,3,4, leaving "15".
`getValue`	Return the current version of the string.
`insert`	Insert a string after the specified location. Example: insert("ABC", 2) in the string "12345" results in "12ABC345".
`replace`	Replace the text at the specified location, inclusive.
`setValue`	Set the value of the string. Erases the previous value.
`hackerSetValue`	Sets the value of the string. Does **NOT** log the change in the undo stack. We will use this to make sure your `undo()` is implemented properly.
`undo`	Undo the last action. Undo is unlimited – you can undo every change (one at a time) all the way back to the initial empty string. The implementation may not store the previous value of the string. **Note:** Storing the full String when we've explicitly said not to is considered cheating and will result in an F in the course.

The trivial way to do this is to use a stack to store the previous value of the string. This works for trivial data but would cause real-world documents and pictures to increase in size by two orders of magnitudes. It's a really bad idea. Also, **you're not allowed to do it**. It wouldn't be worth 20 points if it was that easy.

You're going to do it the right way. You are going to use something called a Command pattern. When an edit method is called, store the changes (the action the user took) in an Action object. If they undo the operation, you use the data in the action object to undo the change (or have the action modify the text directly). We will **not** tell you what's in the Action object or how it works. There are many ways to do it, from simple data storage to interfaces and superclasses. Pick a way that makes the most sense to you.

---

4. Expression Evaluator (20 points)

Tests: Binary trees
Rank: Medium

Mathematical expressions such as `(8 + 4) * (7 - (6 / 3))` can be represented as binary trees such as:

```
          *
       /     \
      +        -
     / \      / \
    8   4    7   /
                / \
               6   3
```

Leaf nodes are numbers, branch nodes are operations.

Humans prefer to use infix notation (`8 + 4`, with the operator in the middle of the arguments) but that's kind of a pain to turn into a tree so we're going to give you your data in prefix notation. In prefix notation, the operators are in front of the arguments, starting with the operator that's in the middle of the expression (i.e., the one that goes in the root node).

```
* + 8 4 - 7 / 6 3
```

Prefix notation is easier to convert to a binary tree. Also, since i realize how many of you miss scheme, remember that you can read this code as:

```
(* (+ 8 4) (- 7 (/ 6 3)))
```

ExpressionTree
+ root : ExpressionTreeNode
+ ExpressionTree(String prefixExpression) - createExpressionTree(List<String> prefixExpression) : ExpressionTreeNode + getExpressionAsInfix() : String + getExpressionAsPostfix() : String + getExpressionAsPrefix() : String + getStructure() : String + getValue() : float

ExpressionTreeNode
+ data : String + leftChild : ExpressionTreeNode + rightChild : ExpressionTreeNode
+ ExpressionTreeNode(String data)

You might need one or two more methods but we'll let you decide what they are.

`ExpressionTree(`
`   String prefixExpression)`
	The constructor builds the tree from a prefix operation. To keep things simple, all operators will be binary. There will be one space between each item in the string.

`createExpressionTree`	Recursive method to create a tree from an infix string.
`getExpressionAsInfix()`	Return the expression in infix notation. `((8 + 4) * (7 - (6 / 3)))` Just to be safe, put parentheses everywhere, whether needed or not. Spaces around the operator.
`getExpressionAsPostfix()`	Return the expression in postfix notation. `((8 4 +)(7 (6 3 /)-)*)` Spaces after the numbers.
`getExpressionAsPrefix()`	Return the expression in prefix notation. `(*(+ 8 4)(- 7(/ 6 3)))` Spaces before the numbers.
`getStructure()`	The normal method we use for printing.
`getValue()`	Return the answer for the expression.

5. **Tree Search** (10 points)

Tests: Tree node traversal, path finding
Rank: Challenging

How do you find a node in a tree? There are several ways. In this question you need to implement four: breadth-first search, depth-first search, depth-limited depth-first search and iterative deepening depth-first search.

SearchTree
`+ root : SearchTreeNode`

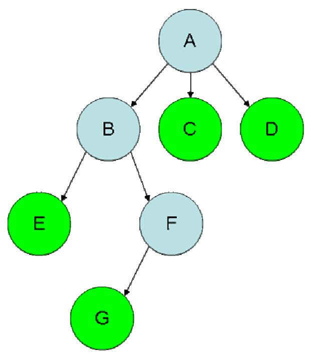
`+ findBFS(String value) : List<String>`
`+ findDFS(String value) : List<String>`
`+ findDL(String value, int limit) : List<String>`
`+ findIDDFS(String value) : List<String>`
`+ getTreeStructure() : String`

SearchTreeNode

`+ data : String`
`+ children : List<SearchTreeNode>`

`+ getChild(int index) : SearchTreeNode`

Each call to a find method should log itself with the profiler (name only, no parentheses or arguments). The unit tests will use the profiler to make sure you're checking the right number of nodes.



`findBFS`	Searches horizontally. To find the node G, breadth-first search would look at A, B, C, D, E, F, G in that order. You are allowed to use your `MyQueue` for this question. Returns an ordered list of Strings, each String containing the data of a node on the search path. For G it would return {A,B,F,G}.
`findDFS`	Searches vertically. To find the node D, depth-first search would look at A, B, E, F, G, C, D in that order. Returns an ordered list of Strings, each String containing the data of a node on the search path. For D it would return {A,D}.
`findDLDFS`	Searches vertically but quits after a certain depth. To find the node D with a depth

	limit of 4, DLDFS would look at A, B, E, F, G, C, D in that order. To find the node D with a depth limit of 2, DLDFS would look at A, B, C, D in that order. With a depth limit of less than 4, DLDFS cannot find the node G. If a node is not found, return `null`.
`findIDDFS`	Searches vertically but quits after a certain depth (here, initial depth limit = 1). If it doesn't find the node, it increases the depth limit by 1 then tries again. To find the node D, IDDFS would look at A then A, B, C, D. To find the node F it would look at A then A, B, C, D then A, B, E, F.
`getChild`	Return the specified child. This method **must** notify the profiler that it was called. Make sure the profiler entry is spelled correctly – "getChild".
`getTreeStructure`	Return a string showing the tree structure.