420-406-AB Data Structures
Winter 2019

April 1st
Worth 4%

Assignment V: Profiling

(Due on Check on Léa)

Profiling is one technique used to analyze the time it takes to run parts of your program. The goal is to figure out how much time is spent in the different sections and identify possible performance *bottlenecks*.

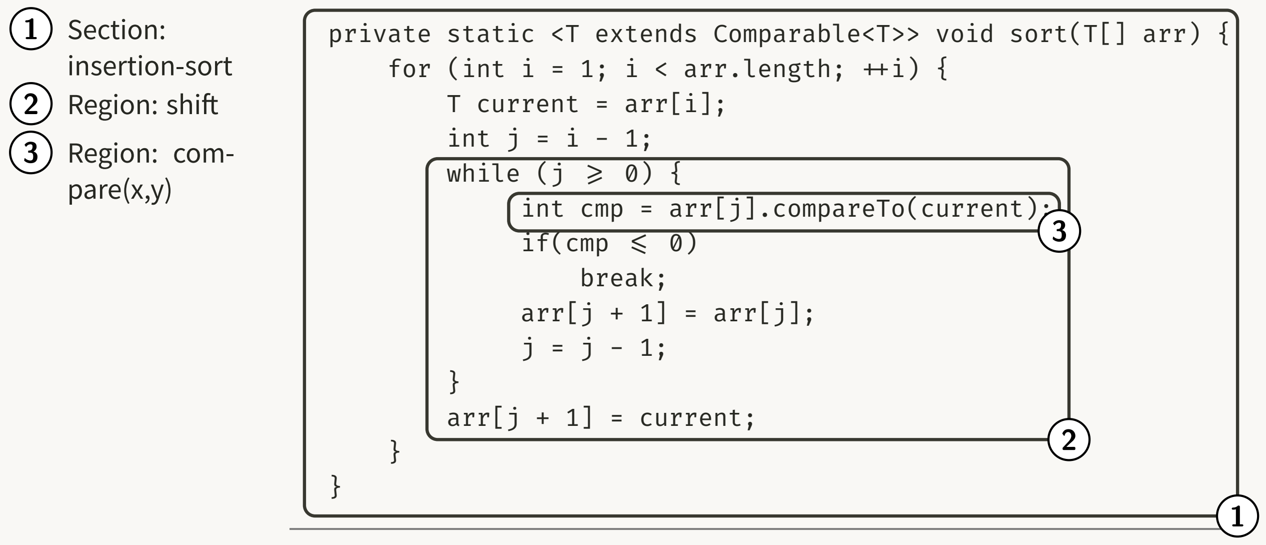
In this assignment, we will write a simple profiler and use it on the list and map data structures we've been developing in the course.

1 Example

Our profiler divides up the program we're testing into two parts: sections and regions.

- Sections are parts of code that we will produce a profiling report on. They have a duration in nanoseconds (ns). Each section should be run only once when profiling.
- Regions are parts of code within a Section that we want specific profiling information on. We want two pieces of information: the number of time the region was executed and the total duration of all these executions in nanoseconds (ns).

Example. Profiling insertion sort:



```
private static <T extends Comparable<T>> void sort(T[] arr) {
    for (int i = 1; i < arr.length; ++i) {
        T current = arr[i];
        int j = i - 1;
        while (j >= 0) {
            int cmp = arr[j].compareTo(current);
            if(cmp <= 0)
                break;
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = current;
    }
}
```

Report. Running the above with 1000 randomly generated `String`s give these results.

Section: Insertion Sort

Region	Run Count	Total Time	Percent of Section	
shift	999	63785.43µs	99.67%	■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
compare(x,y)	255332	33408.76µs	52.21%	■■■■■■■■■■■■■■■
TOTAL	1	63993.96µs	100.00%	■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

The `TOTAL` region is the time data for the entire section. The implementation of the report is given to you as part of the assignment starter.

2 Design

Implement a class `Profiler` to collect and process the profiling data. Use the provided class as a starting point.

2.1 API

Here is the API for our profiler:

Signature	`void startSection(String label)`
Description	Starts a new profiling section.
Pre-conditions	Does not occur inside an existing section.
Post-conditions	A new profiling section is started.

Signature	`void endSection()`
Description	Ends a section.
Pre-conditions	Must be paired with a corresponding call to `startSection( .. )`.
Post-conditions	The current profiling section is ended.

Signature	`void startregion(string label)`
Description	Starts a new profiling region.
Pre-conditions	Occurs within a section.
Post-conditions	A new profiling region is started. Has no effect if the region is not within a section.

Signature	`void endRegion()`
Description	Ends a region.
Pre-conditions	Must be paired with a corresponding call to `startRegion( .. )`.
Post-conditions	The current profiling region is ended. Has no effect if the region is not within a section.

Signature	`List<Section> getSections()`
Description	Generate all the current sections.
Pre-conditions	All started sections and regions have ended.
Post-conditions	Builds and returns a list of all the profiler's current sections. Clears the profiler of the reported sections.

2.2 Implementation

2.2.1 Singleton class: Profiler

Implement the `Profiler` using the "singleton" design pattern, allowing you to add sections and regions anywhere in your application.

Start from the provided `Profiler` class.

2.2.2 Inner class: Mark

The implementation of sections and regions must be relatively quick, so I have setup a simple way to record section and region information using a inner class called `Mark`. Instances of this class are appended to a list as sections and regions are encountered. Please take a look the code in the starter for details.

2.2.3 Class: Section

The representation of a section will be done in a `Section` class. Using the `Report` class as a starting point, design the class `Section`. Do not change the code in `Report`.

2.2.4 Class: Region

The representation of a region will be done in a `Region` class. Using the `Report` class as a starting point, design the class `Region`. Do not change the code in `Report`.

2.2.5 Data structures

Use maps and lists to implement the above classes. Because we are testing our version of theses data structures, it is necessary to use the Java versions instead. They are imported from `java.util.*`. Be sure to import the correct versions in your classes!

3 Profiling our Data Structures

Write a profiling *test suite* that evaluate the efficiency of the list data strucutes `LinkedList` and `ArrayList`, as well as the map data structures `NaiveMap` and `HashMap`.

3.1 Regions

Add the following regions to data structure classes. Be very careful to end every region you start, no matter how you exit the method. You can't include the return statement in your regions, since it's the last line of code in any method.

3.1.1 Lists

Add the following regions:

- Each implementation of `add(x)`, `add(pos,x)`, `remove(pos)`, `get(pos)` and `set(pos,x)`.
- The implementation of `move(n)` in `LinkedList<T>`.
- The implementation of `resize(n)` in `ArrayList<T>`.

3.1.2 Maps

Add the following regions:

- Each implementation of `put(k,v)`, `get(k,v)`, `containsKey(k)`, `remove(k)`, and `keys()`.
- The implementation of `hash(k)` and `rehash()` in `HashMap<K,V>`.

3.2 Suite

Write the following performance tests using 10000 randomly generated `String`.

3.2.1 Lists

Perform the following tests on `LinkedList<T>` and `ArrayList<T>`. Write private methods for each test using the interface type `List<T>`.

- Construct a list using the append operation `add(x)`.

- Construct a list using the operation `add(0,x)`.
- Construct a list using the operation `add(middle, x)` where `middle` is size/2.
- Construct a list using the operation `add(size-2, x)`.
- Reverse a list by removing from the front and adding at the end.
- Reverse a list by removing from the end and adding at the front.
- Find the maximum value in a list using a traditional for loop.
- Find the maximum value in a list using a traversal.

3.2.2 Maps

Perform the following tests on `NaiveMap<K,V>` and `HashMap<K,V>`. Write private methods for each test using the interface type `Map<K,V>`.

- Construct a map.
- Retrieve values from a map with 50% failure rate.

3.2.3 Example

Here is the setup for the first list test for both `LinkedList<T>` and `ArrayList<T>`:

```
public class Main {

    public static final Generator<String> STRING_GENERATOR;
    public static final Random RANDOM;
    public static final int SAMPLE_SIZE = 10000;

    static {
        RANDOM = new Random();
        STRING_GENERATOR = new SentenceGenerator(new WordGenerator("foo bar baz qux quux quuz corge g
    }

    public static void main(String[] args) {

         Profiler.getInstance().startSection("LinkedList -  Initialize with add(x)");
         listInitializeWithAppend(new LinkedList<>());
         Profiler.getInstance().endSection();

         Profiler.getInstance().startSection("ArrayList -  Initialize with add(x)");
         listInitializeWithAppend(new ArrayList<>());
        Profiler.getInstance().endSection();

        Report.printAllSections(Profiler.getInstance().getSections());

    }

    private static void listInitializeWithAppend(List<String> list) {
        for(int i=0; i<SAMPLE_SIZE; i++)
            list.add(STRING_GENERATOR.generate(RANDOM));
    }
}
```

3.3 Reports

I've written a class `Report` that uses our profiler to output results in a simple ASCII table. Use the static method `Report.printSection( .. )` (see the documentation and the example above).

4 Requirements

Your program *must* meet the following requirements:

1. Your program should be clear and well commented. It must follow the "420-406 Style Guidelines" (on Léa).
2. Your code must work with the provided `Report` class.
3. Your `Profiler` class must implement the API in section 2.1, and be designed using the "singleton" pattern.
4. Complete the profiling suite as described in section 3.2. Methods should be written using interface types `List<T>` and `Map<K,V>` to maximize reusability.
5. Submit using Git by following the instructions (on Léa).