Java Tick 4*

The Game of Life has attracted its fair share of fans over the years, and keen "Game of Lifers" have built large repositories of game board patterns. Keen pattern collectors are often interested in patterns which are unique or special in some way. For example, finding the smallest board which has a particular characteristic. In this exercise you will write a program to compute statistics about the game boards found in the Life Lexicon pattern collection in order to identify patterns of particular interest. The list of patterns you should analyse is available from the course website at:

http://www.cl.cam.ac.uk/teaching/current/ProgJava/life.txt

Your program should be capable of determining the following statistics for each pattern:

1. Number of generations before the pattern enters a cycle
2. Length of cycle (a still-life has a cycle length of 0)
3. Largest growth rate in the number of live cells in successive generations
4. Largest death rate in the number of dying cells in successive generations
5. Largest population of live cells in any generation

Start by copying across your source code from your Tick 4 submission, updating the code so that it is inside package `uk.ac.cam.your-crsid.tick4star` and renaming `LoaderLife` to `StatisticsLife`.

Next, you should create a new class called `Statistics` in a file called `Statistics.java` to store the statistics for a particular pattern. Your `Statistics` class should provide (at least) the following methods:

```
public double getMaximumGrowthRate()
public double getMaximumDeathRate()
public int getLoopStart()
public int getLoopEnd()
public int getMaximumPopulation()
```

You will need to include some private fields to store data, the design of which is left up to you. If l_t represents the number of live cells at time t, then the *growth rate* is defined as $(l_{t+1}-l_t)/l_t$. The method `getMaximumGrowthRate` should return the maximum growth rate between any two consecutive generations of a pattern as a floating point value. For example, if a world has 12 live cells at generation t and 24 live cells at generation t+1, then the growth rate is `1.0`.

If l_t represents the number of live cells at time t, then the *death rate* is defined as $(l_t-l_{t+1})/l_t$.The method `getMaximumDeathRate` should return the maximum death rate between any two consecutive generations as a floating point value. For example, if a world has 24 live cells at generation t and 12 live cells at generation t+1, then the death rate is `0.5`.

The method `getLoopStart` should return the generation at which a pattern cycle begins; `getLoopEnd` should return the generation at which a pattern cycle ends. For example, if a world has the same state at generation 5 and generation 15, `getLoopStart` should return 5 and `getLoopEnd` should return 14. The method `getMaximumPopulation` should return the largest number of live cells found in any generation.

Finally, you should update your implementation of `StatisticsLife`. The `StatisticsLife` class should iterate through all the patterns found in a file or at a URL provided as the first argument on the command line. For each pattern, your implementation should create a new instance of `Statistics` and update the state of the statistics object so that the methods defined above for objects of type `Statistics` return the correct results for the given pattern. To create instances of type `Statistics` from a specific pattern, your implementation of `StatisticsLife` should include a new method with the following prototype:

```
public static Statistics analyse(Pattern p)
```

Since your implementation of `StatisticsLife` will take some time to run, you should print out the string "`Analysing`" followed by the name of the pattern currently under analysis so that you can check progress. Once all the patterns in the given file or URL have been analysed, your implementation should print a blank line followed by the names of the patterns which have: (1) the largest number of generations before the start of the loop, (2) the longest cycle, (3) the biggest growth rate, (4) the biggest death rate, and (5) the largest population. You should include the values which these patterns have in parentheses. Here is an example:

```
crsid@machine:~> java uk/ac/cam/your-crsid/tick4star/StatisticsLife Patterns.txt
Analysing PatternName1
Analysing PatternName2
Analysing PatternName3
Analysing PatternName4
Analysing PatternName5
Analysing PatternName6
Analysing PatternName7

Longest start: PatternName1 (12)
Longest cycle: PatternName3 (8)
Biggest growth rate: PatternName3 (2.3)
Biggest death rate: PatternName7 (0.7)
Largest population: PatternName6 (23)
```

You should test your program on files containing your own patterns, as well as the set of patterns provided on the course website. Once you are happy your program works correctly, construct a jar file named `crsid-tick4star.jar` containing your code and a file `output.txt` containing the entire output of your program when run with the URL http://www.cl.cam.ac.uk/teaching/current/ProgJava/life.txt and email it as an attachment to `ticks1a-java@cl.cam.ac.uk`. Your jar file should contain:

```
META-INF/
META-INF/MANIFEST.MF
uk/ac/cam/your-crsid/tick4star/output.txt
uk/ac/cam/your-crsid/tick4star/Pattern.java
uk/ac/cam/your-crsid/tick4star/Pattern.class
uk/ac/cam/your-crsid/tick4star/PatternFormatException.java
uk/ac/cam/your-crsid/tick4star/PatternFormatException.class
uk/ac/cam/your-crsid/tick4star/PatternLoader.java
uk/ac/cam/your-crsid/tick4star/PatternLoader.class
uk/ac/cam/your-crsid/tick4star/Statistics.java
uk/ac/cam/your-crsid/tick4star/Statistics.class
uk/ac/cam/your-crsid/tick4star/StatisticsLife.java
uk/ac/cam/your-crsid/tick4star/StatisticsLife.class
```

You should set the entry point of the jar file so that the following works:

```
crsid@machine:~> java -jar crsid-tick4star.jar \
http://www.cl.cam.ac.uk/teaching/current/ProgJava/life.txt
```

Optional: the Game of Life is supposed to be played on an infinite game board rather than one of a fixed size. Extend your tests to work out which patterns are being artificially altered by a finite fixed-size game board. For example, you might like to consider the Octogon oscillator (not affected) and the Glider pattern (affected).