Java Tick 2*

Much of the investigation into the Game of Life and other cellular automata considers patterns with particular properties such as infinite growth or oscillatory behavior. For this exercise you will write another implementation of TinyLife called LoopingLife. LoopingLife should determine if the pattern of cells found at generation t has been seen before at, say, generation t-n, thus detecting a loop or repeating pattern.

Since the game world is finite, all starting configurations will necessarily repeat at some point (they may stabilize to a static configuration which is simply a loop with period 1). The period of the loop in the Game of Life may be quite large, so in this tick you will search for a loop within at most 100 generations before giving up.

Begin by taking a copy of `TinyLife.java` and `PackedLong.java` files which you produced for Tick 2 and move them to a new package called `uk.ac.cam.your-crsid.tick2star`. Delete the `play` method and replace it with a method called `findLoop` which should have the same method prototype (recall from Workbook 2 that a method prototype describes the number and the types of the arguments and return types of a method). Our search algorithm will proceed by keeping a record of all the worlds which we have previously generated. When generation t of the world is calculated, you will need to check the history of all previous worlds to see if the grid pattern associated with the current generation has been seen before; if the pattern has occurred previously (at, say, generation t-n) then we have found a loop of period n.

You should store the history of previous generations in a Java array. You will explore Java arrays in more detail next week, but for now you only need to know that an array of `long` of length n has the capacity to store n values of type `long`. The first element of the array is at index 0 and the final element is at index n-1.

To create a new array of type `long` of length 100 you should write:

```
long[] myArray = new long[100];
```

Every element of an array of primitive values is initialized to a default value—for a `long` array the default value is `0L`. To retrieve or update the value stored at position `i` in an array you use square brackets. For example

```
long value = myArray[i]; //retrieve the value stored at position i
myArray[i] = 1000L;       //store the value 1000 at position i
myArray[i]++;            //increment the value stored at position i
```

You should now implement the body of the `findLoop` method. Your implementation should use an array of type `long` to store the history of up to 100 generations. Your program should record a history of generations into the array and print out the string "i to j" if a loop is found between generation i and j. One way to print out a message of this form is:

```
System.out.println(i + " to " + j);
```

(In this code snippet Java will convert the two integers `i` and `j` to strings automatically and concatenate the three strings together before sending the final string to the terminal for display.)

If no loops are found after 100 generations you should print "No loops found". Once you are satisfied with your program you should put it in a jar file named `crsid-tick2star.jar` and submit it to `ticks1a-java@cl.cam.ac.uk`. Your jar file should contain the following files and directories:

```
META-INF/
META-INF/MANIFEST.MF
uk/ac/cam/your-crsid/tick2star/LoopingLife.class
uk/ac/cam/your-crsid/tick2star/LoopingLife.java
uk/ac/cam/your-crsid/tick2star/PackedLong.java
uk/ac/cam/your-crsid/tick2star/PackedLong.class
```

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

```
crsid@machine:~> java -jar crsid-tick2star.jar 0x1824428181422418
0 to 4
crsid@machine:~>
```

(The previous example is a loop with a periodicity of 5. The longest oscillation the Lecturers know about within an 8-by-8 world has a periodicity of 32. If you find a longer loop, let us know. Or, if you can prove why loops of greater periodicity are not possible we would be even more interested!)