tick2 submission from James Wood

Name	James Wood (jdw74)
College	ROBIN
Submission contents	uk/ac/cam/jdw74/tick2/PackedLong.java uk/ac/cam/jdw74/tick2/TinyLife.java uk/ac/cam/jdw74/tick2/answers.txt
Ticker	UNKNOWN
Ticker signature	

PackedLong.java

```java
package uk.ac.cam.jdw74.tick2;

public class PackedLong {

    /*
    * Unpack and return the nth bit from the packed number at index position;
    * position counts from zero (representing the least significant bit)
    * up to 63 (representing the most significant bit).
    */
    public static boolean get(long packed, int position) {
        // set "check" to equal 1 if the "position" bit in "packed" is set to 1
        long check = packed >> position & 1L;
        return (check == 1L);
    }

    /*
    * Set the nth bit in the packed number to the value given
    * and return the new packed number
    */
    public static long set(long packed, int position, boolean value) {
        if (value) {
            packed |= 1L << position;
            // update the value "packed" with the bit at "position" set to 1
        }
        else {
            packed &= ~(1L << position);
            // update the value "packed" with the bit a "position" set to 0
        }
        return packed;
    }
}
```

TinyLife.java

```java
package uk.ac.cam.jdw74.tick2;

class TinyLife {
    public static void print(long world) {
        System.out.println("-");
        for (int row = 0; row < 8; row++) {
            for (int col = 0; col < 8; col++) {
                System.out.print(getCell(world, col, row) ? "#" : "_");
            }
            System.out.println();
        }
    }

    public static boolean getCell(long world, int col, int row) {
        return 0 <= col && 0 <= row && col < 8 && row < 8 ?
            PackedLong.get(world, row * 8 + col) : false;
    }

    public static long setCell(long world, int col, int row, boolean value) {
        return 0 <= col && 0 <= row && col < 8 && row < 8 ?
            PackedLong.set(world, row * 8 + col, value) : world;
    }

    public static int countNeighbours(long world, int col, int row) {
        return
            (getCell(world, col - 1, row - 1) ? 1 : 0)
          + (getCell(world, col    , row - 1) ? 1 : 0)
          + (getCell(world, col + 1, row - 1) ? 1 : 0)
          + (getCell(world, col - 1, row    ) ? 1 : 0)
          + (getCell(world, col + 1, row    ) ? 1 : 0)
          + (getCell(world, col - 1, row + 1) ? 1 : 0)
          + (getCell(world, col    , row + 1) ? 1 : 0)
          + (getCell(world, col + 1, row + 1) ? 1 : 0);
    }

    // Skeleton looks awful
    public static boolean computeCell(long world, int col, int row) {
        int count = countNeighbours(world, col, row);
        return count == 3 || (getCell(world, col, row) && count == 2);
    }

    /*public static boolean computeCell(long world,int col,int row) {

        // liveCell is true if the cell at position (col,row) in world is live
        boolean liveCell = getCell(world, col, row);

        // neighbours is the number of live neighbours to cell (col,row)
        int neighbours = countNeighbours(world, col, row);

        // we will return this value at the end of the method to indicate whether
        // cell (col,row) should be live in the next generation
        boolean nextCell = false;

        //A live cell with less than two neighbours dies (underpopulation)
        if (neighbours < 2) {
            nextCell = false;
        }

        //A live cell with two or three neighbours lives (a balanced population)
        else if (liveCell && (neighbours == 2 || neighbours == 3))
            nextCell = true;

        //A live cell with with more than three neighbours dies (overcrowding)
        else if (neighbours > 3)
            nextCell = false;

        //A dead cell with exactly three live neighbours comes alive
        if (neighbours == 3)
            nextCell = true;

        return nextCell;
    }*/

    public static long nextGeneration(long world) {
        long nextWorld = 0;
        for (int col = 0; col < 8; col++)
            for (int row = 0; row < 8; row++)
                nextWorld = setCell(nextWorld, col, row,
                                    computeCell(world, col, row));
        return nextWorld;
    }

    public static void play(long world) throws Exception {
        int userResponse = 0;
        while (userResponse != 'q') {
            print(world);
            userResponse = System.in.read();
            world = nextGeneration(world);
        }
    }

    public static void main(String[] args) throws Exception {
        play(Long.decode(args[0]));
    }
}
```

answers.txt

```
- 1
row * 8 + col
- 2
i % 8
- 3
i / 8
- 4
1
- 5
0
- 6
world              |col|row|result
-------------------|---|---|------
0x20A0600000000000L| 0 | 0 |
0x20A0600000000000L| 0 | 0 |
0x20A0600000000000L| 0 | 0 |
0x20A0600000000000L| 0 | 0 |
0x20A0600000000000L| 0 | 0 |
```