```java
/**
 *Elliot Duncan
 *Horton 7th
 *5/11/24
 *
 *@(#)World.java
 *
 * Statically stores the mutable game grid, and runs the interations for
 *stepping through generations. all methods are static, since there should only
 *ever be one world, so all references to world reference the one.
 */

import java.awt.Rectangle;
import java.util.Arrays;
import java.util.stream.Collectors;

public class World {
  private static int WORLD_SIZE;
  private static int[][] world;
  private static int generation;
  private static String seedName;
  /**
   * returns the size of the world in cells
   * @return the size of the world
   */
  public static int getWORLD_SIZE() { return WORLD_SIZE; }

  /**
   * gets the name of the seed that was used to initialize the world.
   * @return name of the seed
   */
  public static String getSeedName() { return seedName; }

  /**
   * Returns the bounding box of the current layout of the world in the form of
   * a rectangle that represents the box.
   * @return the bounding box
   */
  public static Rectangle getBoundingBox() {

    int minX = Integer.MAX_VALUE;
    int maxX = Integer.MIN_VALUE;
    int minY = Integer.MAX_VALUE;
    int maxY = Integer.MIN_VALUE;
    for (int i = 0; i < WORLD_SIZE; i++) {
      for (int j = 0; j < WORLD_SIZE; j++) {
        if (world[i][j] > 0) {
          minX = Math.min(j, minX);
          maxX = Math.max(j, maxX);
          minY = Math.min(i, minY);
```

```
        maxY = Math.max(i, maxY);
      }
    }
  }
  return new Rectangle(minX, minY, 1 + maxX - minX, 1 + maxY - minY);
}

/**
 * Creates the contents of a file to store the current world in the form of a
 * *.rle file which is able to then be loaded by this program.
 * @return the encoding
 */
public static String saveWorldEncoding() {
  int firstCol = (int)getBoundingBox().getX();
  String encoding = "";
  for (int[] i : world) {
    String s = "";
    for (int j : i) {
      s += j;
    }
    if (s.indexOf("1") >= 0) {
      s = (s.substring(firstCol, s.lastIndexOf('1') + 1));

      String[] rows = (s.split("(?<=(.))(?!\\1)"));
      for (String g : rows) {
        char c = (g.charAt(0) == '0') ? 'b' : 'o';
        String l = (g.length() == 1) ? "" : "" + g.length();
        g = l + c;
        encoding += g;
      }
      encoding += '$';
    }
  }

  encoding = encoding.substring(0, encoding.length() - 1) + '!';
  encoding = "#N " + seedName + "\n#C Generation " + generation + "\n" +
             String.format("x = %.0f, y = %.0f, rule = B3/S23\n",
                           getBoundingBox().getWidth(),
                           getBoundingBox().getHeight()) +
             encoding;
  return encoding;
}

/**
 * Returns a 2d array of ints which represents the state of the world and its
 * cells.
 * @return the world
 */
public static int[][] getWorld() { return world; }
```

```
/**
 * Returns the current generation of the world.
 * @return the current generation;
 */
public static int getGeneration() { return generation; }

/**
 * processes all of the information to generate the next state of the world.
 * First calculates the number of neighbors that every cell has, and stores
 * that in a new 2d array. If a cell has 0 or 1 neighbors it dies, 2 or 3 and
 * it lives on, 4 or more and it dies, and any dead cell with exactly 3 live
 * neighbors also becomes alive.
 */
public static void nextGeneration() {
  generation++;
  int[][] n = new int[WORLD_SIZE][WORLD_SIZE];

  for (int r = 0; r < WORLD_SIZE; r++) {
    for (int c = 0; c < WORLD_SIZE; c++) {

      final int RIGHT = (c + 1) % WORLD_SIZE;
      final int DOWN = (r + 1) % WORLD_SIZE;
      final int LEFT = (c - 1 < 0) ? WORLD_SIZE - (c + 1) : c - 1;
      final int UP = (r - 1 < 0) ? WORLD_SIZE - (r + 1) : r - 1;

      n[r][c] = world[UP][LEFT] + world[UP][c] + world[UP][RIGHT] +
                world[DOWN][LEFT] + world[DOWN][c] + world[DOWN][RIGHT] +
                world[r][LEFT] + world[r][RIGHT];
    }
  }

  for (int row = 0; row < n.length; row++) {
    for (int col = 0; col < n[row].length; col++) {
      if (n[row][col] < 2 || n[row][col] > 3)
        world[row][col] = 0;
      if (n[row][col] == 3 && world[row][col] == 0)
        world[row][col] = 1;
    }
  }
}
/**
 * sets the current state of the world to the specified seed.
 * @param s the seed to add
 */
public static void setWorld(Seed s) {
  seedName = s.getName();
  generation = 0;
  int[][] se = s.getCells();
  WORLD_SIZE = Math.max(Math.max(s.getSizeY() * 3, s.getSizeX() * 3), 128);
  world = new int[WORLD_SIZE][WORLD_SIZE];
```

```
    for (int i = 0; i < s.getSizeY(); i++)
      for (int j = 0; j < s.getSizeX(); j++)
        world[i + WORLD_SIZE / 3][j + WORLD_SIZE / 3] += se[i][j];
  }

  /**
   * overloaded version of the other setWorld, which takes a filename and turns
   * it into a seed first.
   * @param filePath the filepath
   */
  public static void setWorld(String filePath) { setWorld(new Seed(filePath)); }

  /**
   *Returns the string form of the world, as a graphical representation with '#'
   *representing a live cell.
   *@return the string
   */
  @Override
  public String toString() {
    return Arrays.asList(world)
                .parallelStream()
                .map(s -> {
                  return Arrays.stream(s)
                       .mapToObj(c -> { return (c == 1) ? "#" : " "; })
                       .collect(Collectors.joining(""));
                })
                .collect(Collectors.joining("\n")) +
        "\n " + generation;
  }
}
```