tick4 submission from James Wood

Name	James Wood (jdw74)
College	ROBIN
Submission contents	uk/ac/cam/jdw74/tick4/ExceptionTest.java uk/ac/cam/jdw74/tick4/LoaderLife.java uk/ac/cam/jdw74/tick4/Pattern.java uk/ac/cam/jdw74/tick4/PatternFormatException.java uk/ac/cam/jdw74/tick4/PatternLife.java uk/ac/cam/jdw74/tick4/PatternLoader.java uk/ac/cam/jdw74/tick4/Repeat.java
Ticker	UNKNOWN
Ticker signature	

ExceptionTest.java

```
 0  package uk.ac.cam.jdw74.tick4;
 1
 2  class ExceptionTest {
 3      public static void main(String[] args) {
 4          System.out.print("C");
 5          try {
 6              a();
 7          } catch (Exception e) {
 8              System.out.print(e.getMessage());
 9          }
10          System.out.println("A");
11      }
12
13      public static void a() throws Exception {
14          System.out.print("S");
15          b();
16          System.out.print("J");
17      }
18
19      public static void b() throws Exception {
20          System.out.print("T");
21          if (1+2+3==6)
22              throw new Exception("1");
23          System.out.print("V");
24      }
25  }
```

LoaderLife.java

```
 0  package uk.ac.cam.jdw74.tick4;
 1
 2  import java.util.List;
 3
 4  class LoaderLife {
 5      public static void print(boolean[][] world) {
 6          System.out.println("-");
 7          for (int row = 0; row < world.length; row++) {
 8              for (int col = 0; col < world[row].length; col++) {
 9                  System.out.print(getCell(world, col, row) ? "#" : "_");
10              }
11              System.out.println();
12          }
13      }
14
15      public static boolean getCell(boolean[][] world, int col, int row) {
16          return 0 <= row && row < world.length &&
17                 0 <= col && col < world[row].length ?
18              world[row][col] : false;
19      }
20
21      public static void setCell(boolean[][] world, int col, int row, boolean value) {
22          if (0 <= row && row < world.length &&
23              0 <= col && col < world[row].length)
24              world[row][col] = value;
25      }
26
27      public static int countNeighbours(boolean[][] world, int col, int row) {
28          return
29              (getCell(world, col - 1, row - 1) ? 1 : 0)
30            + (getCell(world, col    , row - 1) ? 1 : 0)
31            + (getCell(world, col + 1, row - 1) ? 1 : 0)
32            + (getCell(world, col - 1, row    ) ? 1 : 0)
33            + (getCell(world, col + 1, row    ) ? 1 : 0)
34            + (getCell(world, col - 1, row + 1) ? 1 : 0)
35            + (getCell(world, col    , row + 1) ? 1 : 0)
36            + (getCell(world, col + 1, row + 1) ? 1 : 0);
37      }
38
39      public static boolean computeCell(boolean[][] world, int col, int row) {
40          int count = countNeighbours(world, col, row);
41          return count == 3 || (getCell(world, col, row) && count == 2);
42      }
43
44      public static boolean[][] nextGeneration(boolean[][] world) {
45          boolean[][] nextWorld = new boolean[world.length][world[0].length];
46          for (int row = 0; row < world.length; row++)
47              for (int col = 0; col < world[row].length; col++)
48                  setCell(nextWorld, col, row,
49                          computeCell(world, col, row));
50          return nextWorld;
51      }
52
53      public static void play(boolean[][] world) throws Exception {
54          int userResponse = 0;
55          while (userResponse != 'q') {
56              print(world);
57              userResponse = System.in.read();
58              world = nextGeneration(world);
59          }
60      }
61
62      public static void main(String[] args) throws Exception {
63          if (args.length == 0) {
64              System.out.println("No argument given");
65              return;
66          }
67
68          List<Pattern> ps;
69          String path = args[0];
70          if (path.contains("://"))
71              ps = PatternLoader.loadFromURL(path);
```

```
72          else
73              ps = PatternLoader.loadFromDisk(path);
74
75          if (args.length == 1) {
76              int i = 0;
77              for (Pattern p : ps) {
78                  System.out.println(Integer.toString(i) + ") " + p.toString());
79                  i++;
80              }
81          }
82          else if (args.length == 2)
83              try {
84                  Pattern p = ps.get(Integer.parseInt(args[1]));
85                  boolean[][] world = new boolean[p.getHeight()][p.getWidth()];
86                  p.initialise(world);
87                  play(world);
88              }
89              catch (IndexOutOfBoundsException | NumberFormatException e) {
90                  System.out.println("Second argument is not a valid index");
91              }
92          else {
93              System.out.println("Too many arguments");
94              return;
95          }
96      }
97  }
```

Pattern.java

```
 0  package uk.ac.cam.jdw74.tick4;
 1
 2  import java.text.ParseException;
 3
 4  public class Pattern {
 5
 6      private String name;
 7      private String author;
 8      private int width;
 9      private int height;
10      private int startCol;
11      private int startRow;
12      private String cells;
13
14      public String getName() { return name; }
15      public void setName(String x) { name = x; }
16
17      public String getAuthor() { return author; }
18      public void setAuthor(String x) { author = x; }
19
20      public int getWidth() { return width; }
21      public void setWidth(int x) { width = x; }
22
23      public int getHeight() { return height; }
24      public void setHeight(int x) { height = x; }
25
26      public int getStartCol() { return startCol; }
27      public void setStartCol(int x) { startCol = x; }
28
29      public int getStartRow() { return startRow; }
30      public void setStartRow(int x) { startRow = x; }
31
32      public String getCells() { return cells; }
33      public void setCells(String x) { cells = x; }
34
35      public Pattern(String format) throws PatternFormatException {
36          String[] parts = format.split(":");
37          if (parts.length < 7)
38              throw new PatternFormatException("Too few arguments");
39          if (parts.length > 7)
40              throw new PatternFormatException("Too many arguments");
41
42          name = parts[0];
43          author = parts[1];
44          try {
45              width = Integer.parseInt(parts[2]);
46              if (width <= 0) throw new NumberFormatException();
47          }
48          catch (NumberFormatException e) {
49              throw new PatternFormatException(
50                  "Width argument not a positive integer");
51          }
52          try {
53              height = Integer.parseInt(parts[3]);
54              if (height <= 0) throw new NumberFormatException();
55          }
56          catch (NumberFormatException e) {
57              throw new PatternFormatException(
58                  "Height argument not a positive integer");
59          }
60          try {
61              startCol = Integer.parseInt(parts[4]);
62          }
63          catch (NumberFormatException e) {
64              throw new PatternFormatException(
65                  "x coördinate not an integer");
66          }
67          try {
68              startRow = Integer.parseInt(parts[5]);
69          }
70          catch (NumberFormatException e) {
71              throw new PatternFormatException(
```

```
72                  "y coördinate not an integer");
73          }
74          cells = parts[6];
75      }
76
77      public void initialise(boolean[][] world) throws PatternFormatException {
78          String[] rows = cells.split(" ");
79          char[][] values = new char[rows.length][];
80          for (int i = 0; i < rows.length; i++)
81              values[i] = rows[i].toCharArray();
82
83          for (int i = 0; i < values.length; i++)
84              for (int j = 0; j < values[i].length; j++)
85                  if (" 01".indexOf(values[i][j]) == -1)
86                      throw new PatternFormatException(
87                          "Pattern contains characters other than ‘0’, ‘1’ and ‘ ’");
88                  else
89                      world[startRow + i][startCol + j] = values[i][j] == '1';
90      }
91
92      @Override
93      public String toString() {
94          return name + ":" + author + ":" + width + ":" + height + ":" +
95              startCol + ":" + startRow + ":" + cells;
96      }
97  }
```

PatternFormatException.java

```
0   package uk.ac.cam.jdw74.tick4;
1
2   public class PatternFormatException extends Exception {
3       public PatternFormatException(String message) {
4           super(message);
5       }
6   }
```

PatternLife.java

```
 0  package uk.ac.cam.jdw74.tick4;
 1
 2  class PatternLife {
 3      public static void print(boolean[][] world) {
 4          System.out.println("-");
 5          for (int row = 0; row < world.length; row++) {
 6              for (int col = 0; col < world[row].length; col++) {
 7                  System.out.print(getCell(world, col, row) ? "#" : "_");
 8              }
 9              System.out.println();
10          }
11      }
12
13      public static boolean getCell(boolean[][] world, int col, int row) {
14          return 0 <= row && row < world.length &&
15                 0 <= col && col < world[row].length ?
16              world[row][col] : false;
17      }
18
19      public static void setCell(boolean[][] world, int col, int row, boolean value) {
20          if (0 <= row && row < world.length &&
21              0 <= col && col < world[row].length)
22              world[row][col] = value;
23      }
24
25      public static int countNeighbours(boolean[][] world, int col, int row) {
26          return
27              (getCell(world, col - 1, row - 1) ? 1 : 0)
28            + (getCell(world, col    , row - 1) ? 1 : 0)
29            + (getCell(world, col + 1, row - 1) ? 1 : 0)
30            + (getCell(world, col - 1, row    ) ? 1 : 0)
31            + (getCell(world, col + 1, row    ) ? 1 : 0)
32            + (getCell(world, col - 1, row + 1) ? 1 : 0)
33            + (getCell(world, col    , row + 1) ? 1 : 0)
34            + (getCell(world, col + 1, row + 1) ? 1 : 0);
35      }
36
37      public static boolean computeCell(boolean[][] world, int col, int row) {
38          int count = countNeighbours(world, col, row);
39          return count == 3 || (getCell(world, col, row) && count == 2);
40      }
41
42      public static boolean[][] nextGeneration(boolean[][] world) {
43          boolean[][] nextWorld = new boolean[world.length][world[0].length];
44          for (int row = 0; row < world.length; row++)
45              for (int col = 0; col < world[row].length; col++)
46                  setCell(nextWorld, col, row,
47                          computeCell(world, col, row));
48          return nextWorld;
49      }
50
51      public static void play(boolean[][] world) throws Exception {
52          int userResponse = 0;
53          while (userResponse != 'q') {
54              print(world);
55              userResponse = System.in.read();
56              world = nextGeneration(world);
57          }
58      }
59
60      public static void main(String[] args) throws Exception {
61          try {
62              Pattern p = new Pattern(args[0]);
63              boolean[][] world = new boolean[p.getHeight()][p.getWidth()];
64              p.initialise(world);
65              play(world);
66          }
67          catch (PatternFormatException e) {
68              System.out.println(e.getMessage());
69          }
70          catch (ArrayIndexOutOfBoundsException e) {
71              System.out.println("No argument given");
```

```
72          }
73      }
74  }
```

PatternLoader.java

```
 0  package uk.ac.cam.jdw74.tick4;
 1
 2  import java.io.Reader;
 3  import java.io.BufferedReader;
 4  import java.io.IOException;
 5  import java.io.InputStreamReader;
 6  import java.io.FileReader;
 7  import java.util.List;
 8  import java.util.LinkedList;
 9  import java.net.URL;
10  import java.net.URLConnection;
11
12  public class PatternLoader {
13
14      public static List<Pattern> load(Reader r) throws IOException {
15          BufferedReader buff = new BufferedReader(r);
16          List<Pattern> results = new LinkedList<>();
17
18          String line;
19          while ((line = buff.readLine()) != null)
20              try {
21                  results.add(new Pattern(line));
22              }
23              catch (PatternFormatException e) {}
24
25          return results;
26      }
27
28      public static List<Pattern> loadFromURL(String url) throws IOException {
29          URL destination = new URL(url);
30          URLConnection conn = destination.openConnection();
31          return load(new InputStreamReader(conn.getInputStream()));
32      }
33
34      public static List<Pattern> loadFromDisk(String filename)
35          throws IOException {
36          return load(new FileReader(filename));
37      }
38  }
```

Repeat.java

```
 0  package uk.ac.cam.jdw74.tick4;
 1
 2  public class Repeat {
 3      public static void main(String[] args) {
 4          System.out.println(parseAndRep(args));
 5      }
 6
 7      public static String parseAndRep(String[] args) {
 8          if (args.length < 2)
 9              return "Error: insufficient arguments";
10          int n;
11          try {
12              n = Integer.parseInt(args[1]);
13              if (n < 1) throw new NumberFormatException();
14          } catch (NumberFormatException ex) {
15              return "Error: second argument is not a positive integer";
16          }
17
18          int i = n;
19          String r = "";
20          while (i > 1) {
21              r += args[0] + " ";
22              i--;
23          }
24          return r + args[0];
25      }
26  }
```