```java
package patterns;

import java.util.ArrayList;
import java.util.PriorityQueue;

import patterns.shapes.Square;

public class Evolution {
    private PriorityQueue<Pattern> mypatterns;

    public Evolution() {
        mypatterns = new PriorityQueue<Pattern>();
        PriorityQueue<Pattern> starters = new PriorityQueue<Pattern>();
        for (int i = 0; i < 20; i++) {
            System.out.println(starters.size());
            Pattern p = new Pattern();
            p.add(new Square(5));
            starters.add(p);
        }
        for (int i = 0; i < 10; i++)
            mypatterns.add(starters.remove());
    }

    // this is what this does
    public void evolve() {
        int size = mypatterns.size() - 2;
        PriorityQueue<Pattern> updatedpatterns = new PriorityQueue<Pattern>();
        for (int index = 0; index < size; index++) {
            double threshold = Math.random();
            Pattern newpattern = new Pattern();
            newpattern.set(mypatterns.peek());
            if (threshold < 0.75)
                newpattern.add();
            else
                newpattern.remove();
            if (newpattern.compareTo(mypatterns.peek()) >= 0)
                updatedpatterns.add(newpattern);
            else
                updatedpatterns.add(mypatterns.peek());
            mypatterns.remove();

        }
        updatedpatterns.add(new Pattern());
        updatedpatterns.add(new Pattern());
        mypatterns = updatedpatterns;

    }

```

```
49     public String printbest() {
50         return mypatterns.peek().print();
51     }
52 }
53
```

```java
package patterns;

import java.util.ArrayList;

import patterns.shapes.Diamond;
import patterns.shapes.House;
import patterns.shapes.Petal;
import patterns.shapes.Square;
import patterns.shapes.Triangle;

public class Pattern implements Comparable {
    public static ArrayList<String> availableshapes = new ArrayList<String>();
    private int[][] output;
    private ArrayList<Shape> shapes;
    private int[] numshapes; /*
                              * number of each type of shape in this pattern, in order of availableshapes
                              */
    private int totalshapes; /* sum of all numbers in numshapes */
    private ArrayList<int[]> startindices; /* type of shape, start x, start y */

    public Pattern() {
        // initialzie output grid
        output = new int[50][50];
        for (int row = 0; row < 50; row++) {
            for (int col = 0; col < 50; col++) {
                output[row][col] = 0;
            }
        }
        // initialize instance variables
        availableshapes.add("Square");
        availableshapes.add("Diamond");
        availableshapes.add("House");
        availableshapes.add("Petal");
        availableshapes.add("Triangle");
        shapes = new ArrayList<Shape>();
        numshapes = new int[availableshapes.size()];
        totalshapes = 0;
        startindices = new ArrayList<int[]>();

        // add patterns to output grid
        for (int i = 0; i < 10; i++) {
            add();
        }
    }

```

```
46      public String print() {
47          String pattern = "";
48          for (int row = 0; row < output.length; row++) {
49              for (int col = 0; col < output[0].length; col++) {
50                  pattern += output[row][col];
51                  pattern += " ";
52              }
53              pattern += "\n";
54          }
55          return pattern;
56      }
57
58      // altering methods
59      public void add() {
60          // put shape in output and shape array
61          Shape s = shapegenerator((int) (Math.random() * availableshapes.size
    ()), (int) (Math.random() * 31));
62          s.place(getOutput());
63          shapes.add(s);
64          // update numshapes
65          if (s.getType().equals("Square"))
66              numshapes[0] += 1;
67          else if (s.getType().equals("Diamond"))
68              numshapes[1] += 1;
69          else if (s.getType().equals("House"))
70              numshapes[2] += 1;
71          else if (s.getType().equals("Petal"))
72              numshapes[3] += 1;
73          else
74              numshapes[4] += 1;
75          // update totalshapes
76          totalshapes += 1;
77          // update startindecis
78          int[] indeces = new int[2];
79          indeces[0] = s.startrow;
80          indeces[1] = s.startcol;
81          startindices.add(indeces);
82
83      }
84
85      public void add(Shape s) {
86          s.place(getOutput());
87          shapes.add(s);
88          // update numshapes
89          if (s.getType().equals("Square"))
90              numshapes[0] += 1;
91          else if (s.getType().equals("Diamond"))
92              numshapes[1] += 1;
```

```
 93         else if (s.getType().equals("House"))
 94             numshapes[2] += 1;
 95         else if (s.getType().equals("Petal"))
 96             numshapes[3] += 1;
 97         else
 98             numshapes[4] += 1;
 99         // update totalshapes
100         totalshapes += 1;
101         // update startindecis
102         int[] indeces = new int[2];
103         indeces[0] = s.startrow;
104         indeces[1] = s.startcol;
105         startindices.add(indeces);
106     }
107 
108     public void remove() {
109         int index = (int) Math.random() * shapes.size();
110         shapes.get(index).remove(output);
111     }
112 
113     public void set(Pattern p) {
114         output = p.getOutput();
115         shapes = p.getShapes();
116         numshapes = p.getNumShapes();
117         totalshapes = p.getTotalShapes();
118         startindices = p.getStartIndices();
119     }
120 
121     // ranking methods
122     public double rank() {
123         return 0 - this.getSymetry() + this.getComplexity() + 0.75 *
    this.getDiversity();
124     }
125 
126     public int compareTo(Object p) {
127         if (!p.getClass().equals(this.getClass()))
128             throw new RuntimeException("wrong class");
129         if (((Pattern) p).rank() > this.rank())
130             return -1;
131         else if (((Pattern) p).rank() < this.rank())
132             return 1;
133         else
134             return 0;
135     }
136 
137     // assistant methods
138     public String toString() {
139         String returnedstring = "";
```

```
140         for (int row = 0; row < output.length; row++) {
141             for (int col = 0; col < output[0].length; col++) {
142                 returnedstring += output[row][col];
143                 returnedstring += " ";
144             }
145             returnedstring += "\n";
146         }
147         return returnedstring;
148     }
149 
150     public Shape shapegenerator(int index, int size) {
151         if (index == 0)
152             return new Square(size);
153         else if (index == 1)
154             return new Diamond(size);
155         else if (index == 2)
156             return new House(size);
157         else if (index == 3)
158             return new Petal(size);
159         else
160             return new Triangle(size);
161 
162     }
163 
164     public double getSymetry() {
165         return getVS() + getHS();
166     }
167 
168     public double getComplexity() // number of 1s compared to number of 0s;
    65% is max for ones
169     {
170         double totalsquares = output.length + output[0].length;
171         double onesquares = 0;
172         for (int row = 0; row < output.length; row++) {
173             for (int col = 0; col < output[0].length; col++)
174                 if (output[row][col] == 1)
175                     onesquares += 1;
176         }
177         double percentage = 100 * (onesquares / totalsquares);
178         double complexity = 5 + (Math.pow(percentage - 60, 2) / -320);
179         return complexity;
180     }
181 
182     public double getDiversity() {
183         double standev = getStanDev(numshapes);
184         return -standev;
185     }
186 
```

```
187     // assistant to the assistant methods
188     public double getVS() {
189         int vs = 0;
190         for (int row = 0; row < output.length; row++) {
191             for (int col = 0; col < output[0].length / 2; col++)
192                 if (output[row][col] == output[row][output[0].length - col -
    1])
193                     vs += 1;
194         }
195         double percentage = 100 * vs / 25;
196         return 3.5 * percentage / 100;
197     }
198 
199     public double getHS() {
200         int hs = 0;
201         for (int row = 0; row < output.length / 2; row++) {
202             for (int col = 0; col < output[0].length; col++)
203                 if (output[row][col] == output[output.length - row - 1][col])
204                     hs += 1;
205         }
206         double percentage = 100 * hs / 25;
207         return 3.5 * percentage / 100;
208     }
209 
210     public static double getStanDev(int[] nums) {
211         double average = 0;
212         double[] newnums = new double[nums.length];
213         double output = 0;
214         for (int i = 0; i < nums.length; i++) {
215             average += nums[i];
216         }
217         average /= nums.length;
218         for (int i = 0; i < nums.length; i++) {
219             newnums[i] = Math.pow(nums[i] - average, 2);
220         }
221         for (int i = 0; i < newnums.length; i++) {
222             output += newnums[i];
223         }
224         output /= newnums.length;
225         output = Math.pow(output, 0.5);
226         return output;
227     }
228     // getter methods
229 
230     public int[][] getOutput() {
231         return output;
232     }
233 
```

```
234     public ArrayList<Shape> getShapes() {
235         return shapes;
236     }
237 
238     public int[] getNumShapes() {
239         return numshapes;
240     }
241 
242     public int getTotalShapes() {
243         return totalshapes;
244     }
245 
246     public ArrayList<int[]> getStartIndices() {
247         return startindices;
248     }
249 }
```

```java
package patterns;

import patterns.shapes.Diamond;
import patterns.shapes.House;
import patterns.shapes.Petal;
import patterns.shapes.Square;
import patterns.shapes.Triangle;

public class Runner {
    public static String parseArray(int[][] array) {
        String output = "";
        for (int row = 0; row < array.length; row++) {
            for (int col = 0; col < array[0].length; col++) {
                output += array[row][col];
                output += " ";
            }
            output += "\n";
        }
        return output;
    }

    public static void main(String[] args) {
        int[][] grid = new int[25][25];
        for (int row = 0; row < 25; row++) {
            for (int col = 0; col < 25; col++) {
                grid[row][col] = 0;
            }
        }

        Evolution e = new Evolution();
        e.evolve();
        e.evolve();
        e.evolve();
        Pattern p = new Pattern();
        Square s = new Square(5);
        p.add(s);

        System.out.print(p);

    }
}

```

```java
package patterns;

public abstract class Shape {

    protected int[][] borders /* placement indices in pattern */;
    protected int startrow;
    protected int startcol;

    public Shape(int vertices) {
        borders = new int[vertices][2];
        startcol = 0;
        startrow = 0;
        System.out.println(vertices / 4);

    }

    public int[][] getBorders() {
        return borders;
    }

    public int getStartCol() {
        return startcol;
    }

    public int getStartRow() {
        return startrow;
    }

    public void setStartCol(int i) {
        startcol = i;
        borders[0][1] = startcol;
    }

    public void setStartRow(int i) {
        startrow = i;
        borders[0][0] = startcol;
    }

    public void place(int[][] here) {

        updateBorders(here);

    }

    public void remove(int[][] grid) {
        for (int pair = 0; pair < 50; pair++) {
            for (int col = 0; col < 50; col++)
                grid[pair][col] = 0;
```

```
49          }
50      }
51
52      public int goUp(int[][] here, int sizee, int scol, int srow, int bc) {
53          int myy = srow - 1;
54          int bordercount = bc;
55          int siz = sizee;
56          while (myy > -1 && siz > 0) {
57
58              if (here[myy][scol] == 0 && bordercount < borders.length) {
59                  here[myy][scol] = 1;
60                  borders[bordercount][1] = scol;
61                  borders[bordercount][0] = myy;
62                  bordercount += 1;
63                  siz -= 1;
64              }
65              myy -= 1;
66
67          }
68          return myy;
69      }
70
71      public int goRight(int[][] here, int sizee, int scol, int srow, int bc) {
72
73          int myx = scol + 1;
74          int bordercount = bc;
75          int siz = sizee;
76          while (myx < here[0].length && siz > 0) {
77              if (bordercount < borders.length && here[srow][myx] == 0) {
78                  here[srow][myx] = 1;
79                  borders[bordercount][1] = myx;
80                  borders[bordercount][0] = srow;
81                  bordercount += 1;
82                  siz -= 1;
83              }
84              myx += 1;
85
86          }
87          return myx;
88      }
89
90      public int[] goDiagRight(int[][] here, int sizee, int scol, int srow, int
    bc) {
91          int myx = scol + 1;
92          int myy = srow - 1;
93          int bordercount = bc;
94          int siz = sizee;
95          while (myy >= 0 && myx < here[0].length && siz > 0) {
```

```
 96            if (here[myy][myx] == 0) {
 97                here[myy][myx] = 1;
 98                borders[bordercount][1] = myx;
 99                borders[bordercount][0] = myy;
100                bordercount += 1;
101                siz -= 1;
102            }
103            myx += 1;
104            myy -= 1;
105        }
106        int[] output = new int[2];
107        output[0] = myx;
108        output[1] = myy;
109        return output;
110    }
111
112    public int[] goDiagLeft(int[][] here, int sizee, int scol, int srow, int
   bc) {
113        int myx = scol - 1;
114        int myy = srow - 1;
115        int bordercount = bc;
116        int siz = sizee;
117        while (myy >= 0 && myx >= 0 && siz > 0) {
118            if (here[myy][myx] == 0) {
119                here[myy][myx] = 1;
120                borders[bordercount][1] = myx;
121                borders[bordercount][0] = myy;
122                bordercount += 1;
123                siz -= 1;
124            }
125            myx -= 1;
126            myy -= 1;
127        }
128        int[] output = { myx, myy };
129        return output;
130    }
131
132    public abstract void updateBorders(int[][] here);
133
134    public abstract String getType();
135
136}
137
```

```java
package patterns.shapes;

import patterns.Shape;

public class Diamond extends Shape {
    private int size;
    private int[][] myborders;

    public Diamond(int s) {

        super(4 * s);
        size = s;
        myborders = super.getBorders();
    }

    public void updateBorders(int[][] here) /* updates borders and modifies pattern */
    {

        while (super.getStartCol() < size) {
            super.setStartCol((int) (here[0].length * Math.random()));
        }

        while (super.getStartRow() < size) {
            super.setStartRow((int) (here[0].length * Math.random()));
        }

        here[startrow][startcol] = 1;
        int currentx = startcol;
        int currenty = startrow;
        int[] newcor = goDiagRight(here, size, currentx, currenty, 1);
        currentx = newcor[0];
        currenty = newcor[1];
        goDiagLeft(here, size, currentx - 1, currenty + 1, 1 + size);
        newcor = goDiagLeft(here, size, startcol, startrow, 1 + (2 * size));
        currentx = newcor[0];
        currenty = newcor[1];
        goDiagRight(here, size - 1, currentx + 1, currenty + 1, 1 + (3 * size));
    }

    public String getType() {
        return "Diamond";
    }
}

```

```java
package patterns.shapes;

import patterns.Shape;

public class House extends Shape {
    private int size;
    private int[][] myborders;

    public House(int s) {
        super(2 + 4 * s);
        size = s;
        startcol = 0;
        startrow = 0;
        myborders = super.getBorders();
    }

    public void updateBorders(int[][] here) /* updates borders and modifies pattern */
    {
        // initialize startcol and startrow
        while (startcol < size) {
            startcol = (int) (here[0].length * Math.random());
        }

        while (startrow < size)
            startrow = (int) (here.length * Math.random());

        myborders[0][0] = startrow;
        myborders[0][1] = startcol;
        // place 1s in pattern
        here[startrow][startcol] = 1;
        int currentx = startcol;
        int currenty = startrow;
        currenty = goUp(here, 1, currentx, currenty, 1) + 1;
        goDiagRight(here, size, currentx, currenty, 2);
        currentx = goRight(here, 2 * size, startcol, startrow, 2 + size);
        goUp(here, 1, currentx - 1, currenty + 1, 2 + 3 * size);
        goDiagLeft(here, size - 1, currentx - 1, currenty, 3 + (3 * size));

    }

    public String getType() {
        return "House";
    }
}

```

```java
package patterns.shapes;

import patterns.Shape;

public class Petal extends Shape {
    private int size;
    private int[][] myborders;

    public Petal(int s) {
        super(4 * (s + 1));
        size = s;
        startcol = 0;
        startrow = 0;
        myborders = super.getBorders();
    }

    public void updateBorders(int[][] here) /* updates borders and modifies pattern */
    {
        // initialize startcol and startrow
        while (super.getStartCol() < size) {
            super.setStartCol((int) (here[0].length * Math.random()));
        }

        while (super.getStartRow() < size) {
            super.setStartRow((int) (here[0].length * Math.random()));
        }

        here[startrow][startcol] = 1;
        int currentx = startcol;
        int currenty = startrow;
        int[] currentn = goDiagLeft(here, 1, currentx, currenty, 1);
        currentx = currentn[0] + 1;
        currenty = currentn[1];
        currenty = goUp(here, size, currentx, currenty + 1, 2);
        currentn = goDiagRight(here, 1, currentx, currenty + 1, 2 + size);
        currentx = currentn[0] - 1;
        currenty = currentn[1] + 1;
        goRight(here, size, currentx, currenty, 3 + size);
        currentx = goRight(here, size, startcol, startrow, 3 + 2 * size) - 1;
        currentn = goDiagRight(here, 1, currentx, startrow, 3 + 3 * size);
        currentx = currentn[0] - 1;
        currenty = currentn[1] + 1;
        goUp(here, size, currentx, currenty, 4 + 3 * size);

    }

    public String getType() {
```

```
48         return "Petal";
49     }
50 }
51
```

```java
package patterns.shapes;

import patterns.Shape;

public class Square extends Shape {
    private int size;
    private int[][] myborders;

    public Square(int s) {
        super(4 * s);
        size = s;
        startcol = 0;
        startrow = 0;
        borders = new int[size][2];
    }

    public void updateBorders(int[][] here) /* updates borders and modifies pattern */
    {
        myborders = super.getBorders();
        // initialize startcol and startrow
        while (super.getStartCol() < size) {
            super.setStartCol((int) (here[0].length * Math.random()));
        }

        while (super.getStartRow() < size) {
            super.setStartRow((int) (here[0].length * Math.random()));
        }

        here[startrow][startcol] = 1;
        int currentx = startcol;
        int currenty = startrow;
        currenty = goUp(here, size, currentx, currenty, 1);

        goRight(here, size, currentx, currenty + 1, 1 + size);
        currentx = goRight(here, size, startcol, startrow, 1 + (2 * size));
        goUp(here, size - 1, currentx - 1, startrow, 1 + (3 * size));
    }

    public String getType() {
        return "Square";
    }
}

```

```java
package patterns.shapes;

import patterns.Shape;

public class Triangle extends Shape {

    private int size;
    private int[][] myborders;

    public Triangle(int vertices) {
        super(4 * vertices);
        size = vertices;

        myborders = new int[size][2];
    }

    public void updateBorders(int[][] here) /* updates borders and modifies pattern */
    {

        // initialize startcol and startrow
        while (super.getStartCol() < size) {
            super.setStartCol((int) (here[0].length * Math.random()));
        }

        while (super.getStartRow() < size) {
            super.setStartRow((int) (here[0].length * Math.random()));
        }

        // *myborders[0][0]=startrow;
        // *myborders[0][1]=startcol;

        here[startrow][startcol] = 1;
        int currentx = startcol;
        int currenty = startrow;

        int[] currentn = goDiagRight(here, size, startcol, startrow, 1);
        currentx = currentn[0] - 1;
        currenty = currentn[1] + 1;
        currentx = goRight(here, 2 * size, startcol, startrow, 1 + size) - 1;
        goDiagLeft(here, size - 1, currentx, startrow, 1 + 3 * size);
    }

    public String getType() {
        return "Triangle";
    }
}

```