```
   public void doStep() {
      genePool.evolve();
      phenotype.determineFitness(genePool);
      phenotype.select(genePool);
      control.clearMessages();
      control.println(genePool.generation+
           " generations, best fitness = "+phenotype.bestFitness);
   }

   public void reset() {
      control.setValue("Lattice size", 8);
      control.setValue("Population size", 20);
      control.setValue("Recombination rate", 10);
      control.setValue("Mutation rate", 4);
   }

   public static void main(String args[]) {
      SimulationControl.createApp(new GeneticApp());
   }
}
```

Listing 14.11 The GenePool class.

```
package org.opensourcephysics.sip.ch14.genetic;
import java.awt.*;
import org.opensourcephysics.display.*;

public class GenePool implements Drawable {
   int populationNumber;
   int numberOfGenotypes;
   int recombinationRate;
   int mutationRate;
   int genotypeSize;
   boolean[][] genotype;
   int generation = 0;
   Phenotype phenotype;

   public void initialize(Phenotype phenotype) {
      this.phenotype = phenotype;
      generation = 0;
      numberOfGenotypes =
           populationNumber+2*recombinationRate+mutationRate;
      genotype = new boolean[numberOfGenotypes][genotypeSize];
      for(int i = 0;i<populationNumber;i++) {
         for(int j = 0;j<genotypeSize;j++) {
            if(Math.random()>0.5) { // sets genes randomly
               genotype[i][j] = true;
            }
         }
      }
   }

   public void copyGenotype(boolean a[], boolean b[]) { // copy a to b
      for(int i = 0;i<genotypeSize;i++) {
         b[i] = a[i];
      }
   }
```



```
   public void recombine() {
      for(int r = 0;r<recombinationRate;r += 2) {
         // chooses random genotype
         int i = (int) (Math.random()*populationNumber);
         int j = 0;
         do {
            // chooses second random genotype
            j = (int) (Math.random()*populationNumber);
         } while(i==j);
         // random size to recombine
         int size = 1+(int) (0.5*genotypeSize*Math.random());
         // random location
         int startPosition = (int) (genotypeSize*Math.random());
         // index for new genotype
         int r1 = populationNumber+r;
         // index for second new genotype
         int r2 = populationNumber+r+1;
         copyGenotype(genotype[i], genotype[r1]);
         copyGenotype(genotype[j], genotype[r2]);
         for(int position =
              startPosition;position<startPosition+size; position++) {
            int pbcPosition = position%genotypeSize;
            // make new genotypes
            genotype[r1][pbcPosition] = genotype[j][pbcPosition];
            genotype[r2][pbcPosition] = genotype[i][pbcPosition];
         }
      }
   }

   public void mutate() {
      // index for new genotype
      int index = populationNumber+2*recombinationRate;
      for(int m = 0;m<mutationRate;m++) {
         // choose existing genotype
         int n = (int) (Math.random()*populationNumber);
         // random position to mutate
         int position = (int) (genotypeSize*Math.random());
         // copy genotype
         copyGenotype(genotype[n], genotype[index+m]);
         // mutate
         genotype[index+m][position] = !genotype[n][position];
      }
   }

   public void evolve() {
      recombine();
      mutate();
      generation++;
   }

   public void draw(DrawingPanel panel, Graphics g) {
      // draws genotype as string of red or green squares and lists
      // fitness for each genotype
      if(genotype==null) {
         return;
      }
```