```
      if(phenotype.selectedPopulationFitness==null) {
         return;
      }
      int sizeX = Math.abs(panel.xToPix(0.8)-panel.xToPix(0));
      int sizeY = Math.abs(panel.yToPix(0.6)-panel.yToPix(0));
      for(int n = 0;n<populationNumber;n++) {
         int ypix = panel.yToPix(1.5*n)-sizeY;
         for(int position = 0;position<genotypeSize;position++) {
            if(genotype[n][position]) {
               g.setColor(Color.red);
            } else {
               g.setColor(Color.green);
            }
            int xpix = panel.xToPix(position)-sizeX;
            g.fillRect(xpix, ypix, sizeX, sizeY);
         }
         g.setColor(Color.black);
         g.drawString(String.valueOf(
              phenotype.selectedPopulationFitness[n]),
              panel.xToPix(genotypeSize+1), ypix+sizeY);
      }
   }
}
```

Listing 14.12 The Phenotype class.

```
// population of phenotypes (random bond Ising model)
package org.opensourcephysics.sip.ch14.genetic;
public class Phenotype {
   int L;
   int[][][] J; // random bonds
   int[] populationFitness, selectedPopulationFitness;
   int totalFitness;
   int highestEnergy;
   int bestFitness;

   public void initialize() {
      J = new int[L][L][2];
      highestEnergy = 2*L*L; // highest possible energy
      bestFitness = 0;
      for(int i = 0;i<L;i++) {
         for(int j = 0;j<L;j++) {
            for(int bond = 0;bond<2;bond++) {
               if(Math.random()>0.5) {
                  J[i][j][bond] = 1;
               } else {
                  J[i][j][bond] = -1;
               }
            }
         }
      }
   }

   public void determineFitness(GenePool genePool) {
      totalFitness = 0;
      int state[][] = new int[L][L];
```

```
      populationFitness = new int[genePool.numberOfGenotypes];
      for(int n = 0;n<genePool.numberOfGenotypes;n++) {
         for(int i = 0;i<L;i++) {
            // sets up lattice based on genotype
            for(int j = 0;j<L;j++) {
               int position = i+j*L;
               if(genePool.genotype[n][position]) {
                  state[i][j] = 1;
               } else {
                  state[i][j] = -1;
               }
            }
         }
         for(int i = 0;i<L;i++) {
            // compute energy of lattice configuration
            for(int j = 0;j<L;j++) {
               populationFitness[n] -=
                    state[i][j]*(J[i][j][0]*state[(i+1)%L][j]+
                    J[i][j][1]*state[i][(j+1)%L]);
            }
         }
         // fitness > 0; low energy L implies high fitness
         populationFitness[n] = highestEnergy-populationFitness[n];
         totalFitness += populationFitness[n];
      }
   }

   public void select(GenePool genePool) {
      selectedPopulationFitness = new int[genePool.numberOfGenotypes];
      boolean savedGenotype[][] = new
           boolean[genePool.numberOfGenotypes][genePool.genotypeSize];
      for(int n = 0;n<genePool.numberOfGenotypes;n++) {
         genePool.copyGenotype(genePool.genotype[n], savedGenotype[n]);
      }
      for(int n = 0;n<genePool.populationNumber;n++) {
         int fitnessFraction = (int) (Math.random()*totalFitness);
         int choice = 0;
         int fitnessSum = populationFitness[0];
         while(fitnessSum<fitnessFraction) {
            choice++;
            fitnessSum += populationFitness[choice];
         }
         selectedPopulationFitness[n] = populationFitness[choice];
         if(selectedPopulationFitness[n]>bestFitness) {
            bestFitness = selectedPopulationFitness[n];
         }
         genePool.copyGenotype(savedGenotype[choice],
              genePool.genotype[n]);
      }
   }
}
```