```
public void calculate() {
   L = control.getInt("Lattice size");
   lattice.resizeLattice(L, L); // resize lattice
   // same seed will generate same set of random numbers
   random.setSeed(control.getInt("Random seed"));
   double p = control.getDouble("Site occupation probability");
   // occupy lattice sites with probability p
   for(int i = 0;i<L*L;i++) {
      lattice.setAtIndex(i, random.nextDouble()<p ? -1 : -2);
   }
   // first cluster will have color green (value 0)
   clusterNumber = 0;
}

// returns jth neighbor of site s, where j can be 0 (left),
// 1 (right), 2 (down), or 3 (above).  If no neighbor exists
// because of boundary, return -1. Change this method for
// periodic boundary conditions.
int getNeighbor(int s, int j) {
   switch(j) {
   case 0:                              // left
      if(s%L==0) {
         return -1;
      } else {
         return s-1;
      }
   case 1:                              // right
      if(s%L==L-1) {
         return -1;
      } else {
         return s+1;
      }
   case 2:                              // down
      if(s/L==0) {
         return -1;
      } else {
         return s-L;
      }
   case 3:                              // above
      if(s/L==L-1) {
         return -1;
      } else {
         return s+L;
      }
   default:
      return -1;
   }
}

void colorCluster(int initialSite) {
   // cluster sites whose neighbors have not yet been examined
   int[] sitesToTest = new int[L*L];
   int numSitesToTest = 0; // number of sites in sitesToTest[]
   // color initialSite according to clusterNumber
   lattice.setAtIndex(initialSite, clusterNumber);
   // add initialSite to sitesToTest[]
```



```
   sitesToTest[numSitesToTest++] = initialSite;
   while (numSitesToTest > 0) {
      // grow cluster until numSitesToTest = 0
      // get next site to test and remove it from list
      int site = sitesToTest[--numSitesToTest];
      for(int j = 0;j<4;j++) { // visit four possible neighbors
         int neighborSite = getNeighbor(site,j);
            // test if occupied and not yet added to cluster
            if(neighborSite >= 0 &&
                  lattice.getAtIndex(neighborSite) == -1) {
               // color neighborSite according to clusterNumber
               lattice.setAtIndex(neighborSite, clusterNumber);
               // add neighborSite to sitesToTest[]
               sitesToTest[numSitesToTest++] = neighborSite;
            }
         }
      }
   }

   public void reset() {
      control.setValue("Lattice size", 32);
      control.setValue("Site occupation probability", 0.5927);
      control.setValue("Random seed", 0);
      calculate();
   }

   public static void main(String args[]) {
      CalculationControl control =
            CalculationControl.createApp(new PercolationApp());
      }
   }
```

The percolation threshold p_c is defined as the site occupation probability p at which a spanning cluster first appears in an infinite lattice. However, for a finite lattice, there is a nonzero probability of a spanning cluster connecting one side of the lattice to the opposite side for any value of $p > 0$. For small p, this probability is order p^L (see Figure 12.4), and the probability of spanning goes to zero as L becomes large. Hence, for small p and sufficiently large L, only finite clusters exist.

For a finite lattice, the definition of spanning is arbitrary. For example, we can define a spanning cluster as one that (i) spans the lattice either horizontally or vertically; (ii) spans the lattice in a fixed direction, for example, vertically; or (iii) spans the lattice both horizontally and vertically. These spanning rules are based on open (nonperiodic) boundary conditions, which we will use because the resulting clusters are easier to visualize and determine. The criterion for defining $p_c(L)$ for a finite lattice is also somewhat arbitrary. One possibility is to define $p_c(L)$ as the mean value of p at which a spanning cluster first appears. Another possibility is to define $p_c(L)$ as the value of p for which half of the configurations generated at random span the lattice. These criteria will lead to the same extrapolated value for p_c in the limit $L \to \infty$. In Problem 12.1 we will find an estimated value for $p_c(L)$ that is accurate to about 10%. A more sophisticated analysis discussed in Project 12.13 allows us to extrapolate our results for $p_c(L)$ to $L \to \infty$. In Project 12.17 we will discuss the use of periodic boundary conditions to define the clusters.