```
 1  /* Problem 1--Othello
 2     Since the strategy was completely explained, all that really needs to
 3     be done is emulate the gameplay. */
 4
 5  import java.io.*;
 6  import java.util.*;
 7
 8  public class prob1 {
 9
10   private static Scanner in;
11   private static PrintWriter out;
12   private static int cs;
13   private static char[][] Board;
14   private static int pieces;
15
16   public static void main (String[] args) throws Exception {
17
18    in = new Scanner (new File ("prob1.in"));
19    out = new PrintWriter ("prob1.out");
20    cs = 1;
21    while (true) {
22     Initialize ();
23     if (pieces==0) break; //End program when no pieces are entered
24     Play ();
25    }
26    in.close ();
27    out.close ();
28   }
29
30   /* Initialize creates a new board.  Note that it makes a 10x10 board
31      rather than 8x8 and surrounds it with a ring of blanks.  This will
32      make it easier to test for outflanks without going out of bounds. */
33   public static void Initialize () throws Exception {
34
35    Board = new char[10][10];
36    for (int i=0; i < 10; i++)
37     for (int j=0; j < 10; j++)
38      Board[i][j] = ' ';
39    pieces = 0;
40    while (true) {
41     String p = in.next();
42     if (p.equals("DONE")) return;
43     int r = p.charAt(1)-'0'; //process individual piece
44     int c = p.charAt(2)-'0';
45     Board[r][c] = p.charAt(0);
46     pieces++;
47    }
48   }
49
50   /* OutflankCt computes how many pieces can be flipped if placed there*/
51   public static int OutflankCt (int r, int c) {
52
53    int ct = 0;
54    for (int dx = -1; dx <= 1; dx++) //Go through all directions,
55     for (int dy = -1; dy <= 1; dy++) { //horizonal, vertical, diagonal
56      if (dx==0 && dy==0) continue; //except the "zero vector"
57      int i=1;
58      for (;;i++) //go through until you match color or hit a blank
59       if (Board[r+i*dy][c+i*dx]==Board[r][c] ||
60           Board[r+i*dy][c+i*dx]==' ') break;
61      if (Board[r+i*dy][c+i*dx]==Board[r][c]) ct+=i-1;
62     }
63    return ct;
64   }
65
66   /* Flips the pieces after a piece is played */
```

```
 67   public static void Flip (int r, int c) {
 68
 69    for (int dx = -1; dx <= 1; dx++)
 70     for (int dy = -1; dy <= 1; dy++) {
 71      if (dx==0 && dy==0) continue;
 72      int i=1;
 73      for (;;i++)
 74       if (Board[r+i*dy][c+i*dx]==Board[r][c] ||
 75           Board[r+i*dy][c+i*dx]==' ') break; //If a like piece is on the
 76      if (Board[r+i*dy][c+i*dx]==Board[r][c]) //other side, flip it!
 77       for (int j=1; j < i; j++) Board[r+j*dy][c+j*dx] = Board[r][c];
 78     }
 79    }
 80
 81    /* Attempts to place a piece, the "black" variable indicates whether
 82       the piece is black.  Returns whether a move could be made. */
 83    public static boolean PlacePiece (boolean black) {
 84
 85     int max = 0, mr = -1, mc = -1;
 86     for (int r = 1; r <= 8; r++)
 87      for (int c = 1; c <= 8; c++) { //look for empty squares
 88       if (Board[r][c] != ' ') continue; //Place piece temporarily
 89       if (black) Board[r][c] = 'B'; else Board[r][c] = 'W';
 90       int oct = OutflankCt (r,c);
 91       if (oct > max) { //Keep track of the most flips
 92        max = oct;
 93        mr = r;
 94        mc =c;
 95       }
 96       Board[r][c] = ' '; //Remove piece
 97      }
 98     if (max > 0) { //Place piece for real
 99      if (black) Board[mr][mc] = 'B'; else Board[mr][mc] = 'W';
100      Flip (mr,mc); //Flip 'em
101     }
102     return max > 0;
103    }
104
105    /* Play plays the game, placing pieces back and forth until no more
106       pieces can be played. */
107    public static void Play () {
108
109     int bad = 0;
110     boolean black = true; //Black goes first!
111     while (bad < 2) { //Play until neither player can play
112      if (PlacePiece (black)) bad=0; //Good move
113      else bad++;
114      black = !black; //Switch turns
115     }
116     int bct = 0, wct = 0; //Count black and white pieces
117     for (int r=1; r <= 8; r++)
118      for (int c=1; c <= 8; c++)
119       if (Board[r][c]=='B') bct++;
120       else if (Board[r][c]=='W') wct++;
121     out.printf ("Case %d:\r\n",cs++);
122     PrintBoard(); //Output
123     out.printf (
124      "There are %d black pieces and %d white pieces.\r\n\r\n",
125      bct,wct);
126    }
127
128    /* PrintBoard just prints the board to the output file */
129    public static void PrintBoard () {
130
131     for (int r=1; r <= 8; r++) {
132      for (int c=1; c <= 8; c++) {
```

```
133       if (Board[r][c]==' ') out.printf ("_");
134       else out.printf ("%c",Board[r][c]);
135      }
136      out.printf ("\r\n");
137     }
138     out.printf ("\r\n");
139    }
140   }
141
```