問 11　次の Java プログラムの説明及びプログラムを読んで，設問1，2に答えよ。

〔プログラムの説明〕

リバーシゲームで2人のプレーヤの対戦を支援するプログラムである。

リバーシは，8行8列の升からなる盤に，2人のプレーヤが，一方の面が白，他方の面が黒の石を置いていき，盤上の石の個数を競うゲームである。各プレーヤは自分の石の色（白又は黒）をもつ。盤には，初期状態として，図1のとおり中央に白黒2個ずつの石を配置する。升の位置は，列をアルファベット，行を数字で表し，列・行の順に指定する。例えば，図1で最も左上の升はa1，最も右下の升はh8である。

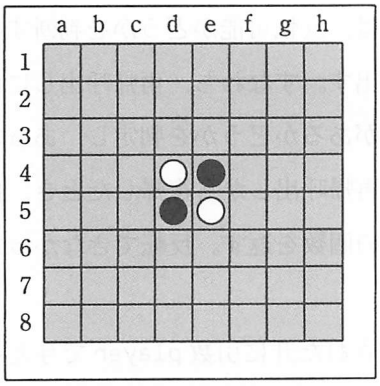

図1　盤の初期状態

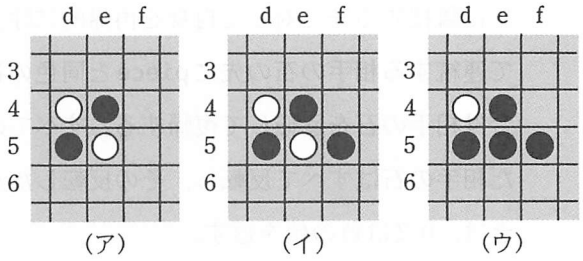

図2　盤に石を置く例

プレーヤは，空いている升に石を置いて，縦，横，斜めのどれかの方向に隣接する相手の石を自分の色の石で挟まなければならない。このとき，挟んだ相手の石をすべて反転させ自分の色の石とする。図2に，黒をもつプレーヤが盤に石を置く例を示す。（ア）は図1の中央部分を抜き出したものである。（イ）は，黒を升f5に置いて，升e5の白を挟んだ状態である。（ウ）は，挟んだ白を反転させて黒にした状態である。相手の石を挟むことができないときは，そのプレーヤは石を置くことができないので，手番をパスする。片方の色の石が盤上からなくなったとき，すべての升が石で埋まったとき，又は空いているどの升にも石が置けなくなったとき，ゲームは終了し，自分の色の石の個数の多い方のプレーヤが勝ちとなる。

列挙 `Piece` は石の面を表す。フィールド `BLACK` は黒，`WHITE` は白を表す。

クラス `Player` は，プレーヤを表す。フィールド `piece` は，そのプレーヤの石の色を表し，フィールド `count` は，盤上での石の個数を表す。

クラスBoardは盤を表し，対戦を支援する次の(1)～(7)の機能を実現する。

(1) 配列gridは，盤の升を表す。配列要素grid[row][col]において，rowは行，colは列である。例えば，升d2は，grid[1][3]で表される。

(2) 列挙Directionは，配列gridにおいて，縦，横，斜め8方向の隣接する升へのrow及びcolのオフセットを表す定数である。

(3) コンストラクタは，インスタンスを生成し，盤を初期状態にする。

(4) メソッドcanPlaceは，引数playerで与えられたプレーヤが石を置ける升があるかどうかを調べ，あるときはtrue，ないときはfalseを返す。

(5) メソッドreverseは，引数row，colで与えられた升に引数pieceで与えられた色の石を置いた場合，その升から引数dirで与えられた方向に隣接する升の石が反転可能であれば反転する。このメソッドは，反転可能かどうかを判断するために隣接する升に対して自身を再帰的に呼び出す。すなわち，再帰呼出しによって連続する相手の石の先にpieceと同色の石があるかどうかを判定し，あった場合は相手の石を反転して復帰する。すべての再帰呼出しから復帰したとき，挟んだ相手の石はすべて反転し，その反転した石の個数を返す。反転できなかったときは，0又は負の値を返す。

(6) メソッドplaceは，引数positionで指定された升に引数playerで与えられたプレーヤの石が置けるかどうかを判定する。石が置ける場合は，升に石を置き，挟んだ相手の石を反転し，反転した石の個数を返す。この処理は，メソッドreverseを呼び出して行う。指定された升に既に石がある場合，又は相手の石を1個も反転できない場合は，0又は負の値を返し，指定された升に石は置かない。

(7) メソッドdisplayは，盤の状態を表示する。

クラスReversiはゲームを実行する。変数playerは，石を置く番のプレーヤである。変数opponentは，playerの対戦相手である。黒をもつプレーヤが常に先手である。playerが石を置いた後に，それぞれのプレーヤの色の石の個数を更新し，ゲームの終了を判断する。ゲームが終了した場合は，盤の最後の状態を表示してプログラムを終了する。終了しなかったときは，playerとopponentが交代し，ゲームを続行する。メソッドpromptは，引数playerの石の色を表示し，石を置く升を“列行”の形式で入力するように促し，入力された升の位置を文字列で返す。

〔プログラム1〕

```
enum Piece { BLACK, WHITE }
```

〔プログラム2〕

```
class Player {
   final Piece piece;
   int count = 2;
   Player(Piece piece) { this.piece = piece; }
   public String toString() { return piece + ": " + count; }
}
```

〔プログラム3〕

```
class Board {
   private final Piece[][] grid = new Piece[8][8];

   private static enum Direction {
      N(-1, 0), NE(-1, 1), E(0, 1), SE(1, 1),
      S(    a    ), SW(1, -1), W(0, -1), NW(-1, -1);
      final int drow, dcol;
      private Direction(int drow, int dcol) {
         this.drow = drow; this.dcol = dcol;
      }
   }

   Board() {
      grid[3][3] = Piece.WHITE; grid[3][4] = Piece.BLACK;
      grid[4][3] = Piece.BLACK; grid[4][4] = Piece.WHITE;
   }

   boolean canPlace(Player player) { /* 実装は省略 */ }

   private int reverse(Piece piece, int row, int col,
                       Direction dir) {
      int nrow = row + dir.drow;
      int ncol = col + dir.dcol;
      Piece next = null;
      if ((nrow >= 0 && nrow < grid.length) &&
          (ncol >= 0 && ncol < grid[nrow].length)) {
         next = grid[nrow][ncol];
      }
      if (next == null)
         return -1;
      if (next == piece)
         return 0;
      int count = reverse(piece, nrow, ncol, dir);
      if (count >= 0) {
         [    b    ] = piece;
         count++;
```

Java

```
        }
        return count;
    }

    int place(Player player, String position) {
        int count = 0;
        try {
            int col = position.charAt(0) - 'a';
            int row = position.charAt(1) - '1';
            if (grid[row][col] != null)
                return -1;
            for (Direction dir : Direction.values()) {
                int n = reverse(player.piece, row, col, dir);
                if (n > 0)
                    count [   c   ] n;
            }
            if (count > 0)
                grid[row][col] = player.piece;
        } catch (IndexOutOfBoundsException e) { }
        return count;
    }

    void display() { /* 実装は省略 */ }
}
```

〔プログラム4〕

```
public class Reversi {
    public static void main(String[] args) {
        Board board = new Board();
        Player player   = new Player(Piece.BLACK),
               opponent = new Player(Piece.WHITE);
        while (true) {
            if (board.canPlace(player)) {
                board.display();
                int n;
                while ((n = board.place(player, prompt(player))) <= 0)
                    System.out.println("wrong position");
                player.count += [   d   ];
                opponent.count -= n;
                System.out.println(player + ", " + opponent);
                if (player.count + opponent.count == 8 * 8
                    || opponent.count == 0) {
                    board.display();
                    break;
                }
            } else if (!board.canPlace(opponent)) {
                board.display();
                break;
            }
            Player p = [   e   ];
```

```
            opponent = player;
            player = p;
        }
    }

    private static String prompt(Player player) { /* 実装は省略 */ }
}
```

設問1　プログラム中の［　　］に入れる正しい答えを，解答群の中から選べ。

aに関する解答群

ア　-1, 0　　イ　-1, 1　　ウ　0, 0

エ　0, 1　　オ　1, 0

bに関する解答群

ア　grid[count][0]　　イ　grid[nrow][col]

ウ　grid[nrow][ncol]　　エ　grid[row][col]

オ　grid[row][ncol]

cに関する解答群

ア　*=　　イ　+=　　ウ　-=

エ　/=　　オ　=

dに関する解答群

ア　++n　　イ　n　　ウ　n + 1

エ　n - 1　　オ　n++

eに関する解答群

ア　board　　イ　opponent　　ウ　Piece.BLACK

エ　Piece.WHITE　　オ　player

設問2　次の表は，プログラム3のメソッド reverse の仕様をまとめたものである。表中の［　　］に入れる正しい答えを，解答群の中から選べ。ここで，b には正しい答えが入っているものとする。

戻り値	意味
正の値	引数 row，col で与えられた升から引数 dir で与えられた方向で反転した石の個数
0	引数 row，col で与えられた升から引数 dir で与えられた方向に［ f ］
-1	引数 row，col で与えられた升から引数 dir で与えられた方向に［ g ］

解答群

ア　隣接する升が存在しない，又は隣接する升に石がない場合

イ　隣接する升に Piece 以外のクラスのインスタンスが存在する場合

ウ　隣接する升の石が引数 piece で与えられた石と同色の場合

エ　隣接する升の石が引数 piece で与えられた石と異なる色の場合

Java