● JKad29X「リバーシ！」

リバーシ（オセロ）を作成せよ。作成方法がわからない場合は以下の手順にしたがって作成していくこと（クラスはJKad29X のみ作成し、手順にしたがってコードの追加・修正を行う）。

手順①「ボードの初期化と表示」

ボードの初期化と表示を行え。作成するメソッドは以下の通り。

作成するメソッド

書式	仕様
public static void initBoard(int[][] board)	配列 board の初期化を行う。 まずはすべての要素を NONE にする。 board[3][3]と board[4][4]は WHITE にする。 board[3][4]と board[4][3]は BLACK にする。
public static void dispBoard(int[][] board)	配列 board の表示を行う。 NONE は「・」、WHITE は「○」（白丸）、BLAKC は「●」（黒丸）。 またボードの上下に列を表す記号「A～H」、 左右に行を表す記号「0～7」を表示する（手順①完成時の画面を参照））

リスト１:「ボードの初期化と表示」

```
public class JKad29X {
    static final int NONE = 0;
    static final int BLACK = 1;
    static final int WHITE = 2;

    public static void main(String[] args) {
        int[][] board = new int[8][8];
        // ボードの初期化
        initBoard(board);
        // ボードの表示
        dispBoard(board);
    }
    // ボードの初期化
    public static void initBoard(int[][] board) {
        作成すること
    }
    // ボードの表示
    public static void dispBoard(int[][] board) {
        String[] strStones = {"・", "●", "○"};
        作成すること
    }
}
```

手順①完成時の画面

```
    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
```

文字と文字の間にタブを入れると
きれいに並ぶ

手順②「とりあえず石を置く」

main メソッドに、ボードの場所（例えば「A0」）を入力すると石を置く処理を作成せよ。黒が先手（白が後手）で交互に石を置くようにすること。なお、ここではボード上にすでに石があるとか、ひっくり返せる石があるかどうかといったルールは無視して、とりあえず指定した場所に石を置くこと（ただしボードの外には置けない）。

追加するメソッド

書式	仕様
public static boolean isInBound(int[][] board, int x, int y)	x と y で指定した場所がボード内にあるかどうか調べる。 ボード内のとき、true を返す。 ボード外のとき、false を返す。

リスト２:「とりあえず石を置く」（太字部分が追加・変更箇所）

```
import java.util.Scanner;

public class JKad29X {
        ⋮
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int[] stoneColor = {BLACK, WHITE};
        String[] strColor = {"黒", "白"};
        int[][] board = new int[8][8];
        // ボードの初期化
        initBoard(board);

        int player = 0;  // 現在のプレイヤー（0：黒、1：白）
        while(true) {
            // ボードの表示
            dispBoard(board);
            // 場所の入力
            作成すること
            // 石を置く
            作成すること
            // プレイヤー交代
            作成すること
        }
        in.close();
    }
        ⋮
    // ボード内にあるかどうかのチェック
    public static boolean isInBound(int[][] board, int x, int y) {
            作成すること
    }
}
```

無限ループを作成し、ボードの表示はループ内へ持って行く。

場所の入力は文字列（「A0」など）で行う。
「exit」が入力されたらループを中断する。

入力された文字列を分解して、横方向の位置（x とする）と縦方向の位置（y とする）を求める。x,y がボード内のとき、そこに現在のプレイヤーの石を置く。ボード外のとき、ループの先頭へ戻ってやり直す。

プレイヤーを交代する。
0（黒）なら1（白）へ、1（白）なら0（黒）へ

手順②完成時の画面

```
    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒の番です。
どこに石を置きますか？（例：A0）＞A0

    A  B  C  D  E  F  G  H
0   ● ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
白の番です。
どこに石を置きますか？（例：A0）＞E3
```

（続き）

```
    A  B  C  D  E  F  G  H
0   ● ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ○ ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒の番です。
どこに石を置きますか？（例：A0）＞A8
そこに石は置けません！
    A  B  C  D  E  F  G  H
0   ● ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ○ ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒の番です。
どこに石を置きますか？（例：A0）＞exit
```

手順③「石をひっくり返す」

相手の石をひっくり返す処理を作成せよ。なお、相手の石を１枚もひっくり返せない場所に、自分の石は置けないようにすること。

追加するメソッド

書式	仕様
public static int turnStone(　int[][] board, 　int x, int y, 　int color)	x、y で指定した場所に color で指定した石を置くことができる場合（同じ color の石で相手の石をはさめる場合）、①はさんだ相手の石をひっくり返して、②ひっくり返した石の枚数を返す（x、y に石を置く処理は main メソッドにて実装済み）。 なお、以下の場合は 0（ひっくり返る石がない）を返す。 ・ボードの外のとき ・すでに石があるとき ・ひっくり返る石がないとき

リスト３:「石をひっくり返す」（太字部分が追加・変更箇所）

```
public class JKad29X {
        ⋮
    public static void main(String[] args) {
            ⋮
        while(true) {
                ⋮
            // 石を置く
            int x = pos.charAt(0) - 'A';        // 横
            int y = pos.charAt(1) - '0';        // 縦
            int color = stoneColor[player];
            if (turnStone(board, x, y, color) > 0) {          //if (isInBound(board, x, y)) {
                board[y][x] = color;
            } else {
                System.out.println("そこに石は置けません！");
                continue;
            }
            System.out.println();
            // プレイヤー交代
                ⋮
        }
            ⋮
    }
        ⋮
    // 石を置いてひっくり返す
    public static int turnStone(int[][] board, int x, int y, int color) {
        作成すること
    }
}
```

条件式を変更する

左上、上、右上、左、右、左下、下、右下のそれぞれの方向に対して、相手の石をひっくり返すことができるかどうか調べる（ひっくり返すことのできる条件を考えること）。ひっくり返すことができるのであれば、ひっくり返す（同時にひっくり返した石の個数をカウントする）。

手順③完成時の画面

```
    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒の番です。
どこに石を置きますか？（例：A0）＞A0
そこに石は置けません！
    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒の番です。
どこに石を置きますか？（例：A0）＞A8
そこに石は置けません！
    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒の番です。
どこに石を置きますか？（例：A0）＞D3
そこに石は置けません！
```

（続き）

```
    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒の番です。
どこに石を置きますか？（例：A0）＞D5
そこに石は置けません！
    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒の番です。
どこに石を置きますか？（例：A0）＞D2

    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ● ・ ・ ・ ・ 2
3   ・ ・ ・ ● ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
白の番です。
どこに石を置きますか？（例：A0）＞C4
    ⋮
```

手順④「仕上げ」

ボード上の石の数を数えるメソッドを追加し、以下の処理を作成せよ。

・「黒」「白」「残り」の数を表示し、すべてのマスが埋まったときにループを抜ける（終了判定）。
・「pass」と入力したらパスする（相手の手番になる）。
・「exit」「pass」以外の文字列が入力されたとき、文字数が 2 文字でないときは再入力させる（ループの先頭へ戻る）。
・ループを抜けたら、「黒」と「白」の石の数を比べて勝敗判定を行う（exit で抜けたときも判定を行う）。

追加するメソッド

書式	仕様
public static int countStone(int[][] board, int color)	ボード上に color の数を返す。

リスト４:「仕上げ」（太字部分が追加・変更箇所）

```
public class JKad29X4 {
        ⋮
    public static void main(String[] args) {
            ⋮
        while(true) {
            // ボードの表示
            dispBoard(board);
            // 石の数の表示と終了判定
            作成すること
            // 場所の入力
                ⋮
            // パス、入力ミスなど
            作成すること
            // 石を置く
                ⋮
            // プレイヤー交代
            player ^= 0x01;
        }
        // 勝敗判定
        作成すること
        in.close();
    }
        ⋮
    // 石の数を数える
    public static int countStone(int[][] board, int color) {
        作成すること
    }
}
```

手順④完成時の画面

```
    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ・ ・ ・ ・ ・ 2
3   ・ ・ ・ ○ ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒：2 個 白：2 個 残り：60 個
黒の番です。
どこに石を置きますか？（例：A0）＞D2

    A  B  C  D  E  F  G  H
0   ・ ・ ・ ・ ・ ・ ・ ・ 0
1   ・ ・ ・ ・ ・ ・ ・ ・ 1
2   ・ ・ ・ ● ・ ・ ・ ・ 2
3   ・ ・ ・ ● ● ・ ・ ・ 3
4   ・ ・ ・ ● ○ ・ ・ ・ 4
5   ・ ・ ・ ・ ・ ・ ・ ・ 5
6   ・ ・ ・ ・ ・ ・ ・ ・ 6
7   ・ ・ ・ ・ ・ ・ ・ ・ 7
    A  B  C  D  E  F  G  H
黒：4 個 白：1 個 残り：59 個
白の番です。
どこに石を置きますか？（例：A0）＞E2
        ⋮
    A  B  C  D  E  F  G  H
0   ● ● ● ● ● ● ● ○ 0
1   ● ○ ○ ○ ○ ○ ○ ○ 1
2   ● ● ○ ● ● ○ ○ ○ 2
3   ● ● ● ○ ○ ○ ○ ○ 3
4   ● ● ○ ● ○ ○ ○ ○ 4
5   ● ○ ○ ● ● ○ ● ● 5
6   ● ・ ○ ○ ● ● ● ● 6
7   ● ・ ○ ○ ○ ○ ○ ○ 7
    A  B  C  D  E  F  G  H
黒：29 個　　白：33 個　　残り：2 個
白の番です。
どこに石を置きますか？（例：A0）＞pass
パスします！
```

（続き）

```
    A  B  C  D  E  F  G  H
0   ● ● ● ● ● ● ● ○ 0
1   ● ○ ○ ○ ○ ○ ○ ○ 1
2   ● ● ○ ● ● ○ ○ ○ 2
3   ● ● ● ○ ○ ○ ○ ○ 3
4   ● ● ○ ● ○ ○ ○ ○ 4
5   ● ○ ○ ● ● ○ ● ● 5
6   ● ・ ○ ○ ● ● ● ● 6
7   ● ・ ○ ○ ○ ○ ○ ○ 7
    A  B  C  D  E  F  G  H
黒：29 個　　白：33 個　　残り：2 個
黒の番です。
どこに石を置きますか？（例：A0）＞B6

    A  B  C  D  E  F  G  H
0   ● ● ● ● ● ● ● ○ 0
1   ● ○ ○ ○ ○ ○ ○ ○ 1
2   ● ● ○ ● ● ○ ○ ○ 2
3   ● ● ● ○ ○ ○ ○ ○ 3
4   ● ● ○ ● ○ ○ ○ ○ 4
5   ● ● ● ● ● ○ ● ● 5
6   ● ● ● ● ● ● ● ● 6
7   ● ・ ○ ○ ○ ○ ○ ○ 7
    A  B  C  D  E  F  G  H
黒：34 個　　白：29 個　　残り：1 個
白の番です。
どこに石を置きますか？（例：A0）＞B7

    A  B  C  D  E  F  G  H
0   ● ● ● ● ● ● ● ○ 0
1   ● ○ ○ ○ ○ ○ ○ ○ 1
2   ● ○ ○ ● ● ○ ○ ○ 2
3   ● ○ ● ○ ○ ○ ○ ○ 3
4   ● ○ ○ ● ○ ○ ○ ○ 4
5   ● ○ ● ○ ● ○ ● ● 5
6   ● ○ ○ ● ● ● ● ● 6
7   ● ○ ○ ○ ○ ○ ○ ○ 7
    A  B  C  D  E  F  G  H
黒：27 個　　白：37 個　　残り：0 個
ゲーム終了です！
白の勝ちです！
```